2017 46th International Conference on Parallel Processing

# Favorable Block First: A Comprehensive Cache Scheme to Accelerate Partial Stripe Recovery of Triple Disk Failure Tolerant Arrays

Luyu Li, Houxiang Ji, Chentao Wu*, Jie Li, and Minyi Guo
Shanghai Key Laboratory of Scalable Computing and Systems,
Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China
*Corresponding Author: wuct@cs.sjtu.edu.cn

***Abstract*—With the development of cloud computing, disk arrays tolerating triple disk failures (3DFTs) are receiving more attention nowadays because they can provide high data reliability with low monetary cost. However, a challenging issue in these arrays is how to efficiently reconstruct the lost data, especially for partial stripe errors (e.g., sector and chunk errors). It is one of the most significant scenarios in practice. However, existing cache strategies are not efficient for partial stripe reconstruction in 3DFTs, which is because the complex relationships among data and parities are usually ignored during the recovery process.**

**To address this problem, in this paper, we proposed a comprehensive cache policy called Favorable Block First (FBF), which can speed up the partial stripe reconstruction of 3DFTs. FBF investigates the relationships among parity chains via allocating various priorities of shared chunks. Thus in the recovery process, by giving higher priorities to the chunks which are shared by more parities chains, FBF can dynamically hold the significant data in buffer cache for partial stripe reconstruction. Obviously, it increases the cache hit ratio and reduces the reconstruction time. To demonstrate the effectiveness of FBF, we conduct several simulations via Disksim. The results show that, compared to typical recovery schemes by combining with classic cache policies (e.g., LRU, LFU and ARC), FBF improves hit ratio by up to $2.47\times$ and accelerates the reconstruction process by $14.90\%$, respectively.**

***Index Terms*—RAID; Erasure Codes; Buffer Cache; Partial Stripe Reconstruction; Parity Chain; Performance Evaluation**

## I. Introduction

With the tremendous growth of data, it is typical to employ numerous number of disks to preserve them in large-scale data centers. The massive usage of inexpensive disks and higher capacity for a single disk increases the probability of data loss or data damage, which affects the reliability of the storage system adversely. Therefore, in cloud storage systems, Redundant Arrays of Inexpensive (or Independent) Disks (RAID), especially disk arrays tolerating triple disk failures (3DFTs), receives more attention than ever [1], [2], [4].

Erasure code is a classic implementation method for triple disk failures (3DFTs). It can be divided into two categories. One class is Maximum Distance Separable (MDS) codes, which aims to offer data protection with optimal storage efficiency. The other class is non-MDS codes, which can improve the performance or reliability by consuming extra storage space. For typical erasure codes employed in 3DFTs such as STAR [5] and Triple-STAR [6], three kinds of parity chains exist, horizontal, diagonal, and anti-diagonal parity chains (introduced in Section II in detail). When disk failure appears, the most frequent case is that a subset of available data and the remaining redundant data (also referred to as "parity") is retrieved by the storage system from the surviving disks.

Partial stripe error is one of the most significant failures to affect the reliability of 3DFTs. For example, latent sector error, as one of the contributors to partial stripe errors, has appeared in 3.45% of studied disks [7]. Besides, partial stripe errors are not independent, and literatures [8] [9] show a high level of spatial and temporal locality. In other words, if a partial stripe error occurs, additional errors are emerged with high probabilities nearby or soon afterwards. Furthermore, the disks with larger capacity are more prone to sector/chunk errors [10]. The inability to either temporarily or permanently access data from certain sectors/chunks can contribute to the excessive mean time to data loss (MTTDL). Such partial stripe errors damage the credibility of storage systems more severely than we expect and encourage us to take an insight to data failure on a more subtle level.

However, with the high probabilities of sector/chunk errors, previous recovery solutions in 3DFTs are not efficient, particularly for partial stripe recovery. There are many reasons. First, existing partial stripe recovery schemes reconstruct the chunks use parity chains in the same direction, which sharply decreases the efficiency of reconstruction process. For example, as introduced in Section II, typical recovery method ignores the fact that some chunks can be shared in multiple chains, and read I/Os is actually larger than needed. Second, if cache replaces the chunks in the stripes whose partial errors are under reconstruction, those evicted chunks have to be fetched again, which significantly influence the efficiency and recovery speed. Therefore proper chunk replacement policy has to be designed for partial stripe error reconstruction, especially when allocated cache size is restricted.

To address this problem, in this paper, we propose a novel cache scheme called FBF, to speed up the partial stripe recovery in 3DFTs. Different from previous cache approaches, FBF gives a higher priority to blocks involved in several parity chains and therefore improves cache hit ratio, especially when cache size is limited for each stripe reconstruction.

The main contributions of this paper are summarized as below,

- We propose a novel cache scheme called FBF, which can increase the cache hit ratio dramatically and speed up the

2332-5690/17 $31.00 © 2017 IEEE
DOI 10.1109/ICPP.2017.31

TABLE I
PARAMETERS AND SYMBOLS USED IN THIS PAPER

| Parameters&Symbols | Description |
|---|---|
| n | number of disks in a disk array |
| P | a prime number for erasure codes |
| $C_{(i,j)}$ | a chunk at the $i$-th row and $j$-th column |
| PriorityDictionary | priority dictionary referenced by cache during reconstruction |
| PartialStripeErrorGroup | a group of partial stripe errors detected in 3DFTs |
| PartialStripeError | error detected on specific stripe |
| RequestedChunk | chunk requried during partial stripe reconstruction |
| ChunkSpace | space allocated in cache for the fetched chunk |
| FetchedChunk | a chunk fetched from disk array |
| Queue3 | queues where chunks' priority is three |
| Queue2 | queues where chunks' priority is two |
| Queue1 | queues where chunks' priority is one |

partial stripe error recovery, especially when cache size is limited.

- We conduct a series of simulations on various cache methods. Compared to existing cache methods, FBF achieves higher hit ratio and speeds up the reconstruction of partial stripe errors in 3DFTs using various erasure codes.

The remainder of the paper is organized as follows. Section II reviews the related work and motivation of our work. Section III illustrates the details of FBF design. Section IV evaluates the performance of our scheme compared with other typical cache methods. Finally we conclude our work in Section V.

## II. RELATED WORK AND MOTIVATION

In this section, we discuss several prevailing erasure codes, cache policies, problems for partial stripe reconstruction in 3DFTs and the motivation of our approach. To simplify the discussion, we summarize the symbols used in this paper in Table I.

### A. Erasure Codes in 3DFTs

Reliability has been a critical issue in storage systems since disk failures and sector errors are typical in large data centers. Erasure coding has been applied in RAID system for a long time. The advantages/disadvantages of erasure coding over simple replication are two-fold. On one hand, it can achieve high reliability with low storage overhead. On the other hand, it requires long reconstruction time. Researchers have presented several erasure codes for triple Disk Failure Tolerant arrays (3DFTs). These codes can be classified into two categories, MDS and non-MDS codes. Typical MDS codes include Reed-Solomon codes [11], Cauchy-RS codes [12], STAR code [5], Triple-Star code [6], Triple-Parity code [15], HDD1 code [14], RSL-code [16], TIP-code[1], EH-Code[37] and RL-code [17], and so on. Typical non-MDS codes contain WEAVER codes [18], HoVer codes [19], T-code [21], Pyramid codes [3], Local Reconstruction Codes [2], Locally Repairable Codes [20], etc.

### B. Typical Cache Replacement Policies

When doing reconstruction, storage systems usually work together with the buffer cache. In order to improve the efficiency of the buffer cache, researchers have proposed many cache replacement algorithms, such as LRU [25], LFU [26], FBR [27], LRU-k [28], 2Q [29], LRFU [30] etc. Cached chunks are examined with different dimensions, like access intervals and access frequencies to determine sacrificial ones. Over the past few decades, researchers pay more attention on the combination of recency and frequency when designing cache algorithms. Besides, novel policies like Victim Disk First (VDF) [23] are proposed recently which take more factor, like miss penalty, into consideration when choosing the eviction during disk reconstruction.

### C. Partial Stripe Errors

There are several types of storage errors that can lead to partial stripe errors, even with multiple layers of checking and correction mechanisms. They can be categorized into two types. The first type is software errors, which is usually difficult to detect and diagnose, such as misdirect write, torn write, data path corruption and parity pollution, etc. Academic studies [32] shows 8.5% of SATA disks would develop silent corruptions and 13% of them are even missed by background verification process. The second type is hardware errors, such as sector errors and chunk errors, which are usually caused by media imperfections and scratches, rotational vibration, read/write head hitting a bump or media and off-track reads or writes etc. [7] These errors are usually repaired by writing recovered data to spare sectors or blocks instead of replacing the whole disk. Partial stripe errors as a critical factor in data reliability. A single partial stripe error can cause data loss when encountered during RAID reconstruction after a disk failure. Lakshmi et al. [7] also find that partial stripe errors like sector errors exhibit a significant amount of spatial and temporal locality. Between 20% to 60% of all errors have a neighbor within a distance of less than 10 sectors in logical sector space. [8]. This probability grows as the disk drive aged.

### D. Problems and Our Motivation

Existing recovering methods of partial stripe error relies on higher level of mechanisms such as RAID reconstruction to obtain the lost data. Once a specific partial stripe error is detected, parity chains are made use of to recover the lost data. However, we find that in 3DFTs arrays, current recovery methods are not efficient enough and usually ignore the fact that several chunks[1] can be involved among multiple parity chains during recovery. In order to accelerate the partial stripe recovery procedure, we proposed a cache scheme called Favorable Block First, which largely reduces the IOs and consequently the mean time to data loss(MTTDL).

[1]In some other research papers [21], chunks are referred to as data blocks as well.

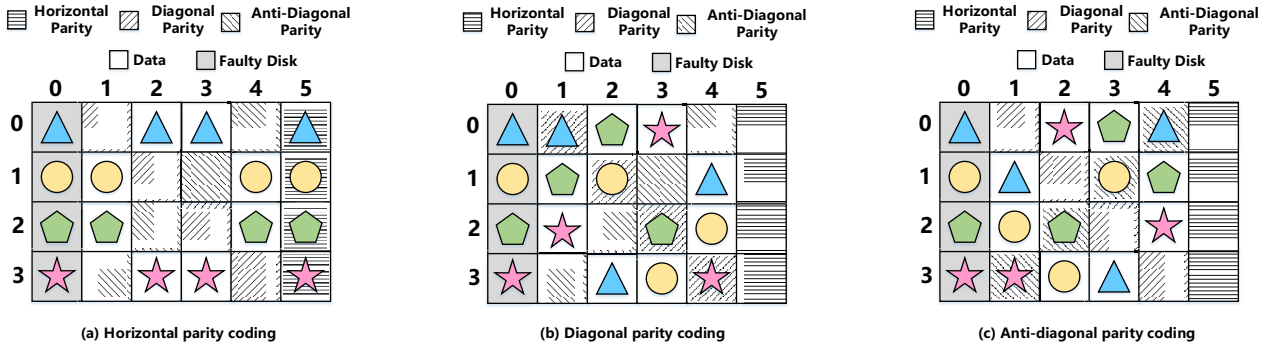


Fig. 1. Encoding of TIP-code ($P = 5$)

Taking TIP-code for an example, Figure 1 illustrates how chunks are encoded for a 6-disk array(P=5). Different shapes are used to indicate various sets of parity chains. For faulty chunks marked with the grey color, there are three ways (Horizontal parity chain, Diagonal parity chain and Anti-diagonal parity chain) for reconstructing each chunk. Typical partial stripe recovery [33] is shown in Figure 2(a). In this figure, parity chains of specific direction are selected to recover all the designated chunks. On the other hand, if the parity chains are intelligently selected (as shown in Figure 2(b)), chunks can be shared by multiple chains, which sharply reduces the total number of chunks needed to be fetched into buffer, together with optimized IOs. Similar approaches are introduced by [22] in order to reconstruct single disk failure.

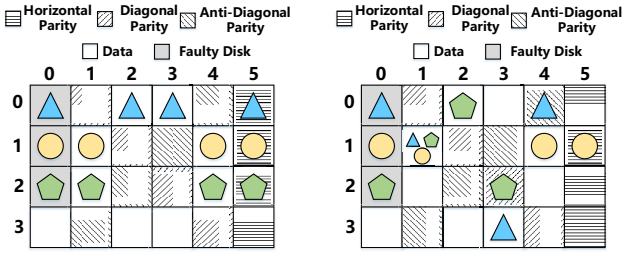


Fig. 2. Example of How Recovery Parity Chains are Selected by Typical and FBF Recovery Scheme with TIP-code ($P = 5$)

Due to the spatial locality of sector/chunk errors [8] [9], partial stripe reconstruction becomes frequent in 3DFTs. When a stripe is under reconstruction, at that time, if the cache evicts the data/parities in the stripe before they are used for the recovery, the efficiency cannot be guaranteed under the reconstruction. So on one hand, related data/parities (e.g., in a same parity chain) should be carefully considered when the cache decide to replace them. On the other hand, due to the limited space of cache, proper selection of chunks for partial stripe recovery is necessary for 3DFTs, especially when the application workload is high.

Another problem for partial stripe recovery is the low efficiency for cache policies. General cache replacement policies

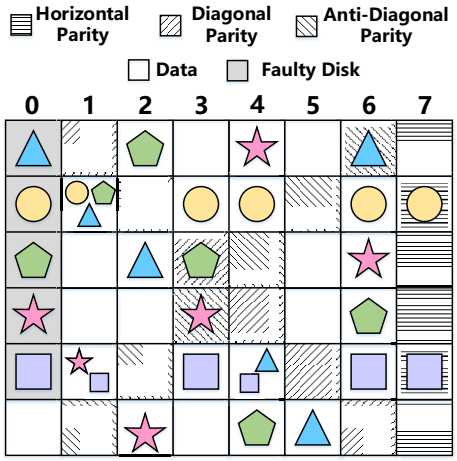


Fig. 3. Example of How Recovery Parity Chains are Selected by FBF Recovery Scheme with TIP-code ($P = 7$)

ignores the special property for partial stripe reconstruction, where shared chunks have to be evicted before the reuse of another parity chain. Consequently, the number of read and write I/Os in practice is much higher than the optimal case. A unique priority-driven cache replacement policy is proposed as well to optimize the whole cache scheme.

The aforementioned situation is more visible in 3DFTs with more disks as Figure 3 indicates. Five sequential chunks are detected with error on $Disk0$, instead of reconstructing faulty chunks by parity chains in the same direction, the scheme makes full use of parities in three directions. However, when applying this recovery scheme in a real environment, one circumstance might appear as Figure 3. Chunks are identified by $(Row\ Number, Column\ Number)$. $Chunk(4,4)$ is fetched to cache when reconstructing $Chunk(0,0)$. When requested for then second time, it had already been evicted due to the large amount of recovery I/Os and limited cache size allocated to reconstruction process. So the cache needs to fetch $Chunk(4,4)$ again from disk arrays. Therefore,

the actual number of I/O is larger than what we expect, so does the reconstruction time. This situation enlarges *the window of vulnerability* (WOV) imperceptibly. Even though cache algorithms like LRU [25], LFU [26], ARC [24] are proved to be effective in industry, there is no cache algorithm designed specifically for this scenario in 3DFTs. Intuitively those chunks shared by multiple parity chains should be given a higher priority in case of being evicted ahead of time. This motivates us to design a cache scheme especially for the scenario of partial stripe errors reconstruction, by combining the high cache performance with optimized I/O cost during reconstruction process via investigating the relationship among various parity chains.

## III. Favorable Block First(FBF) Cache Scheme

In this section, we illustrate the Favorable Block First(FBF) cache scheme in detail, which is an effective cache method to accelerate partial stripe recovery in 3DFTs.

### A. Overview

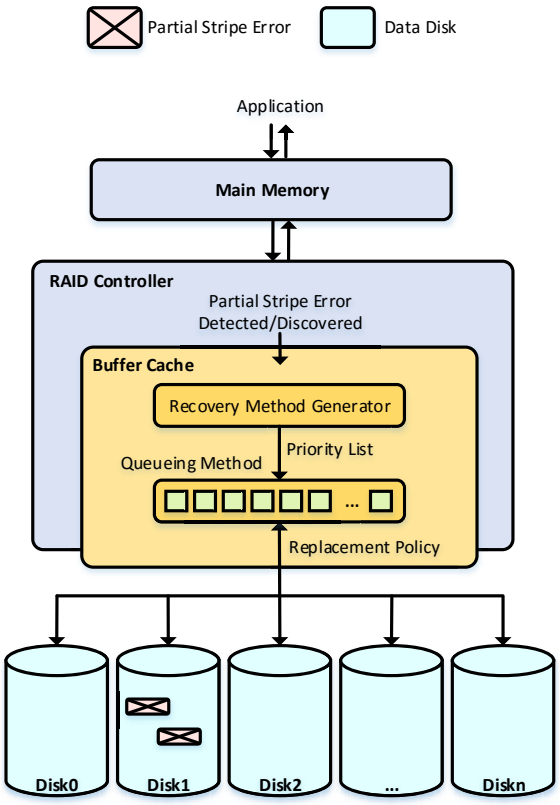


Fig. 4. Overview of the Reconstruction Process in 3DFTs.

A representative reconstruction process of partial stripe recovery of 3DFTs is shown in Figure 4. Favorable Block First(FBF) cache scheme is applied in the RAID Controller, which is transparent to the upper level application. Once partial stripe errors are detected by proactive measure or discovered when particular chunks are accessed, the error information is sent to cache instantly, where lies a Recovery Method Generator, responsible for generating proper recovery scheme for those lost data. Soon afterwards, a list of priorities are created for reference by queuing method. Once designated chunks are fetched from disks, they are cached in the buffer waiting for calculation and reconstruction. Once buffer space is in shortage, replacement policy comes in replacing the chunks with least accessing possibility with new fetched data.

FBF is proposed to improve the hit ratio of cache during partial stripe recovery by issuing a smaller number of disk reads and taking the priority of chunks into consideration. The basic idea of FBF is authorizing chunks with different priorities according to the number of shared parity chains, and the chunks with lower priority have higher chance to be replaced when cache space is insufficient.

FBF abides by the following steps:

**1) Priority Definition:** Generate recovery scheme given failed partial stripe and define priorities for each fetched chunk based on the recovery scheme.

**2) Queueing Method:** Inserting chunks to different queues in cache once fetched from disk arrays.

**3) Replacement Policy:** Replacing chunks with lowest priority first when cache space is not enough.

*1) Priority Definition:* Conventional partial stripe recovery is inefficient and suboptimal because they only use single parity column. In order to make full use of the three parity columns of 3DFTs and create as much overlapped chunks as possible, we generate parity chains by simply looping parity chains of three directions.

According to the layout of 3DFTs, each chunk can be shared by up to three different parity chains, which are horizontal, diagonal, and anti-diagonal parity chains. Once recover scheme is generated, RAID controller can detect how many chains are involved in by setting counter for each chunks in the scheme. Therefore, during the priority definition step, chunks shared by three stripe chains are given the highest priorities, which is because they can decrease the most amount of I/Os. The chunks shared by two stripe chains have the medium priorities, and the chunks referenced once are granted with lowest priorities in FBF. FBF holds the chunks with the highest priorities because (1) the neighbor chunks have high probabilities to be fail due to the spatial locality of sector/chunk errors, (2) the application can access these chunks during partial stripe reconstruction. The priority definition of FBF is summarized in Table II.

In order to define the priorities in a recovery scheme, extra calculation is required. Additional space is needed to store the corresponding priorities of chunks. Even though FBF has temporal and spatial overhead (handling/storing the priorities), the overhead could be negligible due to the following reasons. Firstly, FBF calculates the priorities in advance, which only aims at the partial stripes with various continuous chunk numbers. These priorities can be expanded to all partial stripe

TABLE II
AN SUMMARIZE OF PRIORITY DEFINITION IN FBF

| Priority | Number of Shared Parity Chains | Reduced I/O(s) |
|---|---|---|
| 3 | $\geq$Three | $\leq$ 2 I/Os |
| 2 | Two | $\leq$ 1 I/Os |
| 1 | One | 0 I/Os |

errors with the same format among all disks. Furthermore, these priorities can be enumerated once a same format of partial stripe error is detected again, and no more calculation is required. Hence the temporal cost is quite limited. Secondly, priority is only divided into three types, so two bits are enough to represent priority. Since chunk sizes usually takes at least KB as unit, an attachment of 2 bits for each fetched chunk is trivial.

*2) Queuing:* Since a data can be referenced for up to three times during reconstruction, FBF needs only three queues in recovery. In order to clearly indicate this process, we take recovery scheme in Figure 3 as an example. Similar cache method can be applied to 3DFTs with various codes.

TABLE III
PRIORITY DEFINITION OF RECOVERY SCHEME EXAMPLE USING TIP-CODE(P=7, N=8)

| Priority | Chunks |
|---|---|
| 3 | $C_{(1,1)}$ |
| 2 | $C_{(4,1)}$, $C_{(4,4)}$ |
| 1 | $C_{(0,2)}$,$C_{(0,4)}$,$C_{(0,6)}$,$C_{(1,3)}$,$C_{(1,4)}$,$C_{(1,6)}$,$C_{(1,7)}$,$C_{(2,2)}$,$C_{(2,3)}$, $C_{(2,6)}$,$C_{(3,3)}$,$C_{(3,6)}$,$C_{(4,3)}$,$C_{(4,6)}$,$C_{(4,7)}$,$C_{(5,2)}$,$C_{(5,4)}$,$C_{(5,5)}$ |

After defining priorities for each chunk, according to Figure 3, we can get the priorities of various chunks as shown in Table III. Once chunks are fetched into cache, different priorities are added to the chunks in the corresponding queues. In these queues, various links (referred to as arrows) between chunks are established based on the priorities. It is clearly indicated in Figure 5, where $C_{(1,1)}$ and $C_{(4,4)}$ is attached to $Queue3$ and $Queue2$ individually due to their comparatively high priorities.

When a cache hit occurs, a demoting procedure is started in order to keep the priority queues neatly. Once a chunk in $Queue3$ hits, with a history record of referenced twice in the reconstruction process, it is demoted and inserted to the start point of $Queue2$. As shown in Figure 6, LRU algorithm is used as an example in each queue, the latest accessed data are attached to the end of each queue. Similar to $Queue3$, when the chunk with priority two is hit, it should be demote to $Queue1$ in the same way. During the partial stripe reconstruction, chunks with the lowest priorities have no effectiveness on helping the recovery of another parity chain, they are kept in the lowest cache level and waiting for the I/O requests of applications. Figure 6 shows how $C_{(1,1)}$ is demoted to $Queue2$ and afterwards to $Queue1$ later.

*3) Replacement Policy:* According to the queueing process of FBF, the chunks with different priorities are attached to various queues. The chunks with the high, medium and

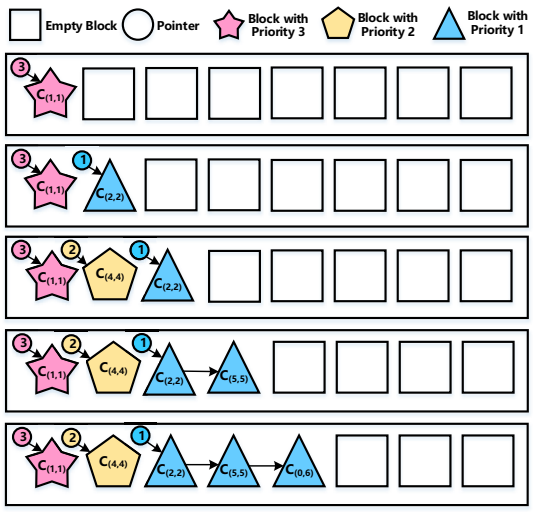


Fig. 5. An Example of Cache Warming Up Process During Reconstruction with requests coming in the sequence of $C_{(1,1)}$, $C_{(2,2)}$, $C_{(4,4)}$, $C_{(5,5)}$, $C_{(0,6)}$

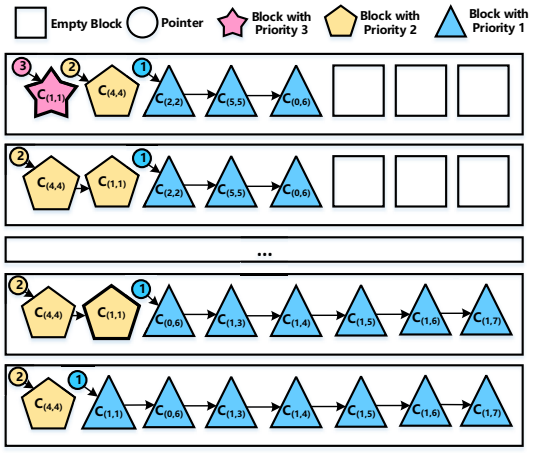


Fig. 6. An Example of Cache Demoting Process During Reconstruction with two $C_{(1,1)}$ requests

low priorities are in $Queue3$, $Queue2$, $Queue1$, respectively. When cache is full, FBF replaces the chunks in $Queue1$ first, and then $Queue2$, and $Queue3$ finally.

Figure 7 illustrates the selection of evicted chunk when a requested chunk comes in. The chunks with dash line indicates they are selected, and is evicted in the following process. In this figure, it can be noticed that when a chunk in $Queue1$ comes in, though the chunks in $Queue2$ (e.g., $C_{(1,1)}$) has not be referenced for a long time, they should be resident in the cache due to their comparatively high priority. Instead, only the chunk with priority 1 is evicted to make space for the incoming chunk.

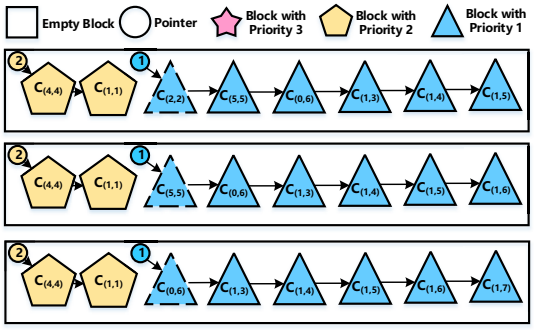


Fig. 7. An Example of Replacement Policy During Reconstruction Process with requests coming in the sequence of $C_{(1,6)}$, $C_{(1,7)}$.

The detailed implementation of FBF is described in Algorithm 1.

### B. Parallel Reconstruction

Disk-Oriented Reconstruction(DOR) and Stripe-Oriented(SOR) Reconstruction [13] are two main parallel disk reconstruction methods. In DOR, $n$ process are created ($n$ is the number of disks in a 3DFTs array) and each process corresponds to one disk. In the scenario, one process is used to handle write operations in spare disk, and the other $n-1$ processes are assigned to read data from surviving disks in parallel. In SOR, multiple processes are created and each processes is responsible for the recovery of several stripes. There is no interaction between any two processes and the stripes are reconstructed simultaneously. Since partial stripe recovery is at the granularity of Chunks, we extends SOR to FBF as well. In this scenario, each process is allocated with a small part of cache, which means the cache size for a single process is limited. By combing with SOR, FBF not only can work for simple reconstruction process in serial, but also does well for parallel reconstruction (e.g., XOR calculations and writes to the spare sector/Chunk simultaneously).

## IV. Evaluation

In this section, we evaluate the performance of partial stripe recovery between Favorable Block First(FBF) and typical cache methods in 3DFTs, which demonstrates the effectiveness of FBF.

### A. Methodology

In this subsection, we give the simulation results to evaluate various cache approaches, which are measured by the metrics defined in the previous subsection.

Disksim 4.0 with default settings is used as the simulator [31] in our evaluation. It is an efficient and highly-configurable disk simulator for a typical storage system. In Disksim, the stripe unit size means the chunk size. Typically, the stripe size is more than 256KB in an array [34], hence chunk size is set to 32KB in our simulations.

**Algorithm 1:** Algorithm of FBF

```
Input: Priority Dictionary, Chunk Read Request
Queue = SearchCache(RequestedChunk)
 // Cache miss
if Queue == NULL then
    if No space in Cache then
        if Queue1 is not empty                    // Replacement policy
         then
            ChunkSpace=Release(Queue1.Pop())
        end
        else if Queue2 is not empty then
            ChunkSpace=Release(Queue2.Pop())
        end
        else if Queue3 is not empty then
            ChunkSpace=Release(Queue3.Pop())
        end
    end
    else
        ChunkSpace = Cache.New()
    end
     // Read from Disk
    FetchedChunk=FetchData(RequestedChunk, ChunkSpace)
    Priority=PriorityDictionary[FetchedChunk]
    switch Prority do
        case 3 do
            Queue3.Attach(FetchedChunk)
        end
        case 2 do
            Queue2.Attach(FetchedChunk)
        end
        case 1 do
            Queue1.Attach(FetchedChunk)
        end
    end
end
 // Cache hit
else
    switch Queue do
        case Queue3 do
            Queue3.Remove(RequestedChunk)
            Queue2.Attach(RequestedChunk)
        end
        case Queue2 do
            Queue2.Remove(RequestedChunk)
            Queue1.Attach(RequestedChunk)
        end
        case Queue1 do
            Queue1.PushToEnd(RequestedChunk)
        end
    end
end
```

The effectiveness of FBF is evaluated using sythetic traces of situations where disks with random size of partial stripes fail. The size of partial stripe errors are set within the range of $[1 \times chunksize, (p-1) \times chunksize]$ on one disk, since chunk is the fundamental recovery unit. While the smallest partial stripe error can happen on only one chunk, it cannot be greater than $(p-1) \times chunksize$ as well, otherwise it would be recovered at the granularity of stripes, which is another topic of 3DFTs reconstruction and solved by previous literatures [22] [36].

Due to the temporal and spatial locality as mentioned in Section II, we assume that the error chunks detected in the same stripe are continuous, and the sizes of partial stripe errors obeys uniform distribution[2], with the average number lies in the half size of the stripe, which is $\frac{p-1}{2} \times chunksize$). In addition, the typical partial stripe recovery scheme utilizes the horizontal parity chains to recover the lost data. In 3DFTs, using vertical (including diagonal/anti-diagonal) parity chains

[2]FBF can be proved under other distributions as well.

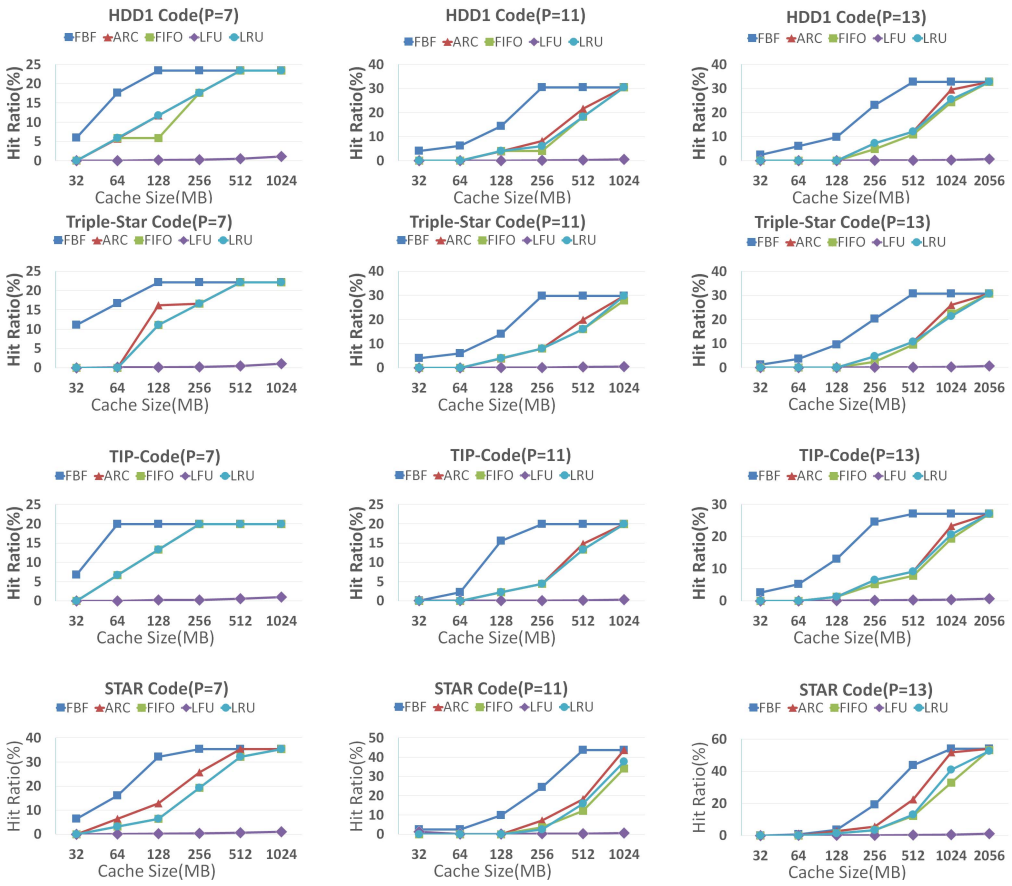


Fig. 8. Cache Hit Ratio During Partial Stripe Reconstruction in 3DFTs Using Various Erasure Codes($P = 7, 11, 13$).

has approximate effects on partial stripe reconstruction, which is not involved in our evaluation.

We Compare FBF with several widely-used cache algorithms, such as FIFO, LRU [25], LFU [26], and ARC [24]. Several popular XOR-based MDS codes in 3DFTs are selected for comparison [3]:

**(1) Codes for $p+1$ disks:** TIP-Code [1], HDD1 [14];

**(2) Codes for $p+2$ disks:** Triple-Star code [6];

**(3) Codes for $p+3$ disks:** STAR code [5].

We selects the following metrics in our evaluation:

**(1) Cache Hit Ratio:** It is the hit ratio of buffer cache when reconstructing partial stripes.

**(2) Total Number of Read Operations During Recovery Process:** It is the total number of read I/Os to recover partial stripes.

**(3) Average Response Time:** It is the average response time for each I/O request to reconstruct partial stripes.

**(4) Reconstruction Time:** It is the total reconstruction time to recover all partial stripes.

[3]Several Reed Solomon based Codes like Local Reconstruction Codes [2] can be applied with FBF as well, by investigating relationships among global/local parity chains during the recovery.

We implement the simulation in a system environment of ubuntu 12.04(32 bit) with Intel Core i5-4430 CPU@3.00GHz and RAM size of 12GB, and start the parallel reconstruction using 128 threads to recover a failed with 1TB capacity. In our simulation, the data access time of buffer cache and data disk are set to 0.5ms and 10ms, respectively.

### B. Results

*1) Cache Hit Ratio:* Cache hit ratio is one of the most important metrics to show the effectiveness of a cache scheme. High cache hit ratio indicates that a larger percentage of requested chunks is hit instead of accessing disk arrays. It can reduce the data access time and accelerate the reconstruction dramatically. The result are shown in Figure 8.

It shows a trend that as cache size increases, the hit ratio keeps increasing and remains stable after cache size exceeds a specific number. This tendency is comprehensible because extremely small cache size always leads to the eviction of valuable chunks and decrease the hit ratio, and this pattern applies to all the cache schemes. Since those chunks being referenced for one time is always missed by cache, the hit ratio reaches a plateau as cache size rises.

Among those graphs in Figure 8, FBF shows an outstanding improvement when cache size is limited, and it reaches plateau

faster than other cache methods. The hit ratio of FBF can be multiple times of other cache methods. For example, in Figure 8, when using HDD1 code with $P = 11$ and $cachesize = 256MB$, the hit ratio of FBF is at least $2.51\times$ than that of other methods. STAR code shows a comparatively higher hit ratio and the possible reason is adjusters of each stripe can be referenced for more than three times and always assigned with highest priority in cache, which can enhance the hit ratio potentially.

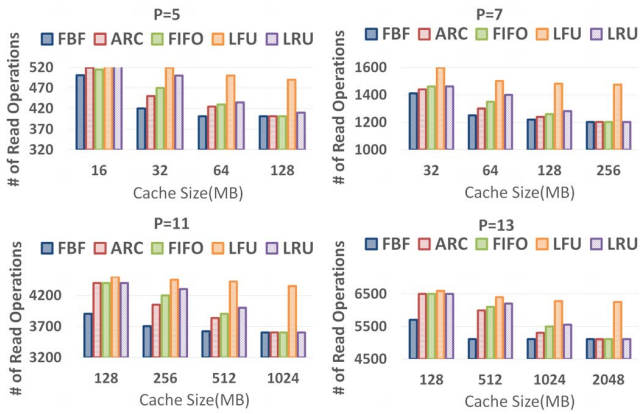


Fig. 9. Number of Read Operations During Partial Stripe Reconstruction in 3DFTs Using TIP-Code($P = 5, 7, 11, 13$).

*2) Number of Read Operations in Disks During Recovery Process:* Due to the limitation of pages, only the number of read operation using TIP-code is shown as an example. Since read operations during partial stripe reconstruction can incredibly affect the total reconstruction time and the reliability of a storage system consequently, we count the number of disk reads during the reconstruction process. The result are presented in Figure 9. It shows that compared with other typical cache methods, FBF can reduce the total read I/O number by up to $22.52\%$(when $prime = 13$ and $cachesize = 512MB$ compared with LFU).

It can be observed that when cache size increases, the number of disk reads decreases since a growing number of chunks are cached instead of being evicted and fetched repeatedly. The number of disk reads is expected to stabilize when cache size is big enough (32 MB when $P = 5$, 64 MB when $P = 7$, 256 MB when $P = 11, 13$). As prime number grows, the stable point of cache size is postponed as well. This tendency owes to the increased stripe size accompanied with the increased prime, which needs larger cache space in order to reach the steady state.In short, FBF shows an obvious advantage when cache size is restricted.

*3) Response Time of Disk Array During Recovery Process:* We study the response time of disk arrays of various erasure codes as shown in Figure 10. The response time of disk arrays plays an important role in the total reconstruction time, and an effective reduction of disk response time can narrow down the Window Of Vulnerability(WOV) as well. In fact, as a general cache scheme for partial stripe recovery of 3DFTs, FBF works well with all the XOR-based 3DFTs erasure codes and results in a relatively shorter response time than all of other cache methods.

In addition to the overall trend that larger cache sizes result in comparatively faster response time, FBF shows distinct superiority compared with other cache methods in the scenario of parity stripe recovery. Taking TIP-code as an example, the maximum reduction is up to $31.39\%$ compared with LFU, $24.51\%$ with FIFO, $24.46\%$ with LRU, $18.02\%$ with ARC when $P = 13$. The advantage of FBF becomes obvious when cache size is limited, but not apparent as cache size keeps growing(when cache size exceeds $2048MB$ in experiments).

*4) Partial Stripe Error Reconstruction Time:* Figure 11 compares cache schemes applying on TIP-code. The reconstruction time decreases as cache size rises, and the time taken by FBF is less than other codes in most of the test cases. Compared with the most general cache method LRU and ARC, FBF have an improvement of up to $14.90\%$ and $12.04\%$, respectively. Even though reconstruction time and response time by FBF shows similar characteristics, the improvement of reconstruction time compared with other cache schemes is less obvious. The reason is the process of reconstruction involves plenty of operations like calculation and writing to spare chunks, which take same time for all the compared cache methods and lessen the statistical percentage of improvement potentially.

TABLE IV
OVERHEAD OF FBF DURING PARTIAL STRIPE RECOVERY IN 3DFTS

| Code | STAR | TripleSTAR | TIP | HDD1 |
|---|---|---|---|---|
| $P = 5$ | | | | |
| Temporal Overhead(ms) | 0.086 | 0.063 | 0.058 | 0.061 |
| Percentage(%) | 1.239 | 1.192 | 1.331 | 1.214 |
| $P = 7$ | | | | |
| Temporal Overhead(ms) | 0.125 | 0.087 | 0.081 | 0.088 |
| Percentage(%) | 1.362 | 1.359 | 1.299 | 1.445 |
| $P = 11$ | | | | |
| Temporal Overhead(ms) | 0.278 | 0.182 | 0.169 | 0.178 |
| Percentage(%) | 2.010 | 1.819 | 1.598 | 1.782 |
| $P = 13$ | | | | |
| Temporal Overhead(ms) | 0.433 | 0.258 | 0.236 | 0.252 |
| Percentage(%) | 2.753 | 2.157 | 2.273 | 2.423 |

*5) Overhead of FBF:* Table IV shows average temporal overhead during partial stripe reconstruction when $P = 5, 7, 11, 13$ with four tested erasure codes. Percentages are calculated as $TemporalOverhead/TotalReconstructionTime$. It is observed that temporal overhead does not have obvious changes as cache size or chunk size increases. The overheads increase a little bit as prime number becomes larger, but they all remains under $2.8\%$, which is acceptable during data reconstruction.

As discussed in SectionIII, only two bits is attached to each fetched chunk as an indicator of priority. Since at least KB is taken as the unit of chunk sizes, the spatial overhead of FBF is negligible during practice.

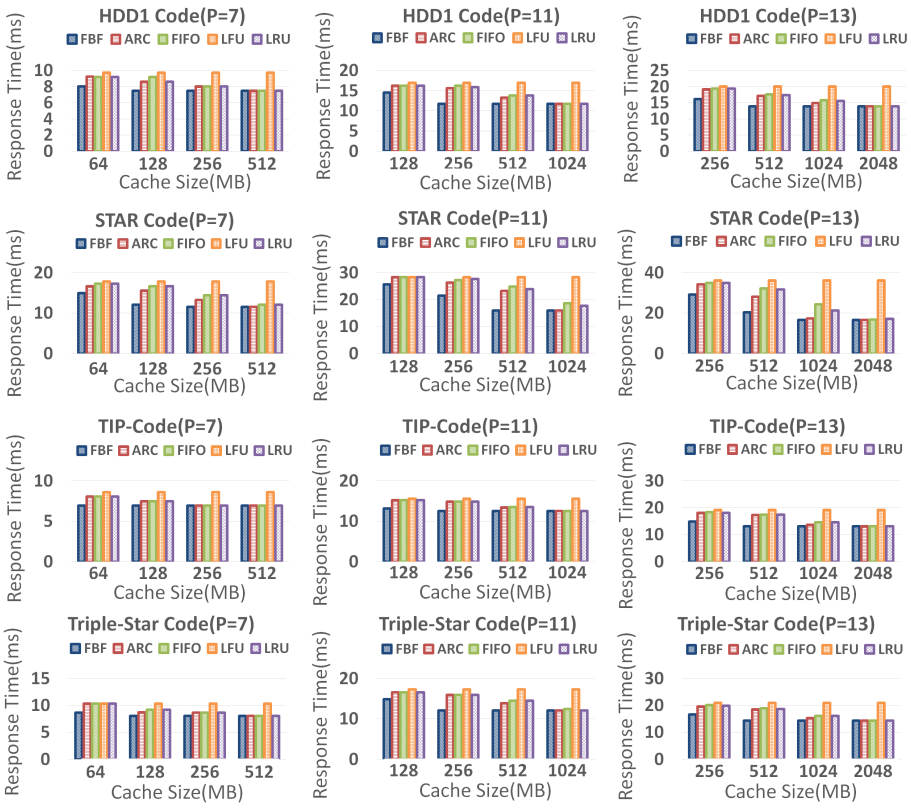


Fig. 10. Average Response Time of Partial Stripe Reconstruction in 3DFTs Using Various Erasure Codes($P = 7, 11, 13$).

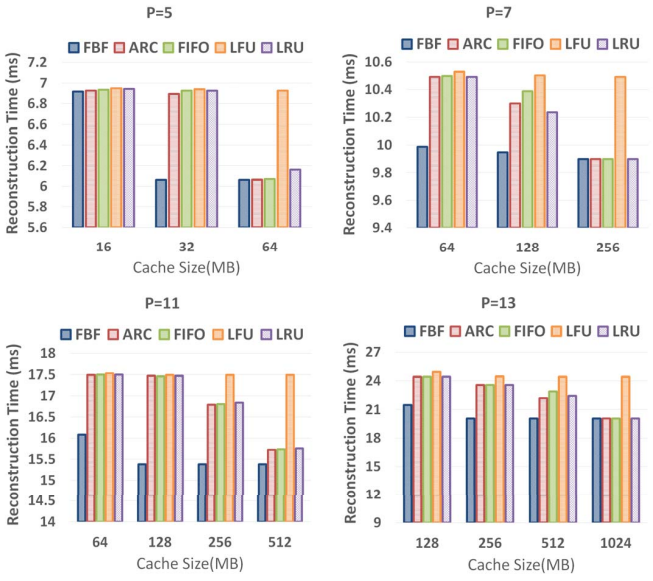


Fig. 11. Partial Stripe Reconstruction Time in 3DFTs Using TIP-Code($P = 5, 7, 11, 13$).

TABLE V
MAXIMUM IMPROVEMENT OF FBF OVER OTHER CACHE POLICIES FOR PARTIAL STRIPE RECOVERY IN 3DFTS

| Cache Management Policy | FIFO | LRU | LFU | ARC |
|---|---|---|---|---|
| Hit ratio | 134.06% | 142.70% | 247.67% | 63.74% |
| Number of reads in disks | 14.13% | 17.14% | 22.52% | 12.37% |
| Response time | 24.51% | 24.46% | 31.39% | 18.02% |
| Reconstruction time | 11.77% | 14.90% | 13.42% | 12.04% |

### C. Analysis

From the results in above subsection, FBF could increase the hit ratio in cache during reconstruction, and reduce the time consumed by referring disk arrays. The maximum improvements of FBF over other cache management policy are summarized in Table V.

There are several reasons to achieve this gains. Firstly, the relationship among parity chains are taken into consideration compared with other general cache methods. Secondly, the favorable chunks are authorized with a higher priority, hence they have less possibilities to be evicted compared with chunks referenced for only once. Thirdly, FBF is a cache scheme utilizing the character of any XOR-based 3DFTs erasure codes, hence the improvement of FBF can apply to a wide range of storage arrays.

## V. Conclusion

In this paper, we proposed a novel cache scheme in order to accelerate partial stripe reconstruction for disk arrays tolerating triple disk failures (3DFTs). By authorizing priorities to data/parity chunks shared by multiple parity chains, FBF can sharply increase the cache hit ratio during reconstruction. Experiment results prove that, FBF improves hit ratio by up to $2.47\times$ and accelerates the reconstruction process by $14.90\%$, respectively. FBF is considered to be effective for parallel and online recovery as well.

## VI. Acknowledgement

We thank anonymous reviewers for their insightful comments. This work is partially sponsored by the the National 863 Program of China (No. 2015AA015302), National 973 Program of China (No.2015CB352403), and the National Natural Science Foundation of China (NSFC) (No. 61572323, and No. 61628208).

## References

[1] Zhang, Y., Wu, C., Li, J. and Guo, M., 2015, June. Tip-code: A three independent parity code to tolerate triple disk failures with optimal update complextiy. In Dependable Systems and Networks (DSN), 2015 45th Annual IEEE/IFIP International Conference on (pp. 136-147). IEEE.

[2] Huang, C., Simitci, H., Xu, Y., Ogus, A., Calder, B., Gopalan, P., Li, J. and Yekhanin, S., 2012, June. Erasure Coding in Windows Azure Storage. In Usenix annual technical conference (pp. 15-26).

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

[4] Tamo, I., Papailiopoulos, D.S. and Dimakis, A.G., 2016. Optimal locally repairable codes and connections to matroid theory. IEEE Transactions on Information Theory, 62(12), pp.6661-6671.

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

[6] Wang, Y., Li, G. and Zhong, X., 2012. 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), pp.1731-1472.

[7] Bairavasundaram, L.N., Goodson, G.R., Pasupathy, S. and Schindler, J., 2007, June. An analysis of latent sector errors in disk drives. In ACM SIGMETRICS Performance Evaluation Review (Vol. 35, No. 1, pp. 289-300). ACM.

[8] Schroeder, B., Damouras, S. and Gill, P., 2010. Understanding latent sector errors and how to protect against them. ACM Transactions on storage (TOS), 6(3), p.9.

[9] Venkatesan, V. and Iliadis, I., 2013, August. Effect of latent errors on the reliability of data storage systems. In Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2013 IEEE 21st International Symposium on (pp. 293-297). IEEE.

[10] Bairavasundaram, L.N., Arpaci-Dusseau, A.C., Arpaci-Dusseau, R.H., Goodson, G.R. and Schroeder, B., 2008. An analysis of data corruption in the storage stack. ACM Transactions on Storage (TOS), 4(3), p.8.

[11] Reed, I.S. and Solomon, G., 1960. Polynomial codes over certain finite fields. Journal of the society for industrial and applied mathematics, 8(2), pp.300-304.

[12] Bloemer, J., Kalfane, M., Karp, R., Karpinski, M., Luby, M. and Zuckerman, D., 1995. An XOR-based erasure-resilient coding scheme.

[13] Zhu, Y., Lee, P.P., Hu, Y., Xiang, L. and Xu, Y., 2012, April. On the speedup of single-disk failure recovery in xor-coded storage systems: Theory and practice. In Mass Storage Systems and Technologies (MSST), 2012 IEEE 28th Symposium on (pp. 1-12). IEEE.

[14] Tau, C.S. and Wang, T.I., 2003, March. Efficient parity placement schemes for tolerating triple disk failures in RAID architectures. In Advanced Information Networking and Applications, 2003. AINA 2003. 17th International Conference on (pp. 132-137). IEEE.

[15] Corbett, P.F. and Goel, A., Netapp, Inc., 2013. Triple parity technique for enabling efficient recovery from triple failures in a storage array. U.S. Patent 8,516,342.

[16] Feng, G.L., Deng, R.H., Bao, F. and Shen, J.C., 2005. New efficient MDS array codes for RAID. Part I. Reed-Solomon-like codes for tolerating three disk failures. IEEE Transactions on Computers, 54(9), pp.1071-1080.

[17] Feng, G.L., Deng, R.H., Bao, F. and Shen, J.C., 2005. New efficient MDS array codes for RAID. Part II. Rabin-like codes for tolerating multiple (/spl ges/4) disk failures. IEEE Transactions on Computers, 54(12), pp.1473-1483.

[18] Hafner, J.L., 2005, December. WEAVER Codes: Highly Fault Tolerant Erasure Codes for Storage Systems. In FAST (Vol. 5, pp. 16-16).

[19] Hafner, J.L., 2006, June. HoVer erasure codes for disk arrays. In Dependable Systems and Networks, 2006. DSN 2006. International Conference on (pp. 217-226). IEEE.

[20] Sathiamoorthy, M., Asteris, M., Papailiopoulos, D., Dimakis, A.G., Vadali, R., Chen, S. and Borthakur, D., 2013, March. Xoring elephants: Novel erasure codes for big data. In Proceedings of the VLDB Endowment (Vol. 6, No. 5, pp. 325-336). VLDB Endowment.

[21] Tang, D., Wang, X., Cao, S. and Chen, Z., 2008, August. A new class of highly fault tolerant erasure code for the disk array. In Power Electronics and Intelligent Transportation System, 2008. PEITS'08. Workshop on (pp. 578-581). IEEE.

[22] Xiang, L., Xu, Y., Lui, J. and Chang, Q., 2010, June. Optimal recovery of single disk failure in RDP code storage systems. In ACM SIGMETRICS Performance Evaluation Review (Vol. 38, No. 1, pp. 119-130). ACM.

[23] Wan, S., Cao, Q., Huang, J., Li, S., Li, X., Zhan, S., Yu, L., Xie, C. and He, X., 2011, June. Victim disk first: an asymmetric cache to boost the performance of disk arrays under faulty conditions. In Proceedings of the 2011 USENIX conference on USENIX annual technical conference (pp. 13-13). USENIX Association.

[24] Megiddo, N. and Modha, D.S., 2003, March. ARC: A Self-Tuning, Low Overhead Replacement Cache. In FAST (Vol. 3, pp. 115-130).

[25] Mattson, R.L., Gecsei, J., Slutz, D.R. and Traiger, I.L., 1970. Evaluation techniques for storage hierarchies. IBM Systems journal, 9(2), pp.78-117.

[26] Aho, A.V., Denning, P.J. and Ullman, J.D., 1971. Principles of optimal page replacement. Journal of the ACM (JACM), 18(1), pp.80-93.

[27] Robinson, J.T. and Devarakonda, M.V., 1990. Data cache management using frequency-based replacement (Vol. 18, No. 1, pp. 134-142). ACM.

[28] O'neil, E.J., O'neil, P.E. and Weikum, G., 1993. The LRU-K page replacement algorithm for database disk buffering. ACM SIGMOD Record, 22(2), pp.297-306.

[29] Johnson, T. and Shasha, D., 1994. X3: A low overhead high performance buffer management replacement algorithm. In Proceedings of the 20th VLDB Conference.

[30] Lee, D., Choi, J., Kim, J.H., Noh, S.H., Min, S.L., Cho, Y. and Kim, C.S., 1996. LRFU (least recently/frequently used) replacement policy: A spectrum of block replacement policies. IEEE Transactions on Computers, 50(12), pp.1353302-1361.

[31] Bucy, J.S., Schindler, J., Schlosser, S.W. and Ganger, G.R., 2008. The disksim simulation environment version 4.0 reference manual (cmu-pdl-08-101). Parallel Data Laboratory, p.26.

[32] Eddy, Josh, Silent data corruption in SATA arrays: A solution, NEC. 2009.

[33] Patterson, D.A., Gibson, G. and Katz, R.H., 1988. A case for redundant arrays of inexpensive disks (RAID) (Vol. 17, No. 3, pp. 109-116). ACM.

[34] Wu, C., He, X., Wu, G., Wan, S., Liu, X., Cao, Q. and Xie, C., 2011, June. Hdp code: A horizontal-diagonal parity code to optimize i/o load balancing in raid-6. In Dependable Systems & Networks (DSN), 2011 IEEE/IFIP 41st International Conference on (pp. 209-220). IEEE.

[35] Kuratti, A. and Sanders, W.H., 1995, April. Performance analysis of the RAID 5 disk array. In Computer Performance and Dependability Symposium, 1995. Proceedings., International (pp. 236-245). IEEE.

[36] Khan, O., Burns, R.C., Plank, J.S., Pierce, W. and Huang, C., 2012, February. Rethinking erasure codes for cloud file systems: minimizing I/O for recovery and degraded reads. In FAST (p. 20).

[37] Jiang, Y., Wu, C., Li, J. and Guo, M., 2015, November. EH-Code: An extended mds code to improve single write performance of disk arrays for correcting triple disk failures. In International Conference on Algorithms and Architectures for Parallel Processing (pp. 34-49). Springer International Publishing.