EC-Fusion: An Efficient Hybrid Erasure Coding Framework to Improve Both Application and Recovery Performance in Cloud Storage Systems

Han Qiu[1], Chentao Wu[1]*, Jie Li[1], Minyi Guo[1], Tong Liu[2], Xubin He[2], Yuanyuan Dong[3], and Yafei Zhao[3]
[1]Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China
[2]Department of Computer and Information Sciences, Temple University, Philadelphia, United States
[3]Alibaba Group, Hangzhou, China
*Corresponding Author: wuct@cs.sjtu.edu.cn

***Abstract*—Nowadays erasure coding is one of the most significant techniques in cloud storage systems, which provides both quick parallel I/O processing and high capabilities of fault tolerance on massive data accesses. In these systems, triple disk failure tolerant arrays (3DFTs) is a typical configuration, which is supported by several classic erasure codes like Reed-Solomon (RS) codes, Local Reconstruction Codes (LRC), Minimum Storage Regeneration (MSR) codes, etc. For an online recovery process, the foreground application workloads and the background recovery workloads are handled simultaneously, which requires a comprehensive understanding on both two types of workload characteristics. Although several techniques have been proposed to accelerate the I/O requests of online recovery processes, they are typically unilateral due to the fact that the above two workloads are not combined together to achieve high cost-effective performance.**

To address this problem, we propose Erasure Codes Fusion (EC-Fusion), an efficient hybrid erasure coding framework in cloud storage systems. EC-Fusion is a combination of RS and MSR codes, which dynamically selects the appropriate code based on its properties. On one hand, for write-intensive application workloads or low risk on data loss in recovery workloads, EC-Fusion uses RS code to decrease the computational overhead and storage cost concurrently. On the other hand, for read-intensive or frequent reconstruction in workloads, MSR code is a proper choice. Therefore, a better overall application and recovery performance can be achieved in a cost-effective fashion. To demonstrate the effectiveness of EC-Fusion, several experiments are conducted in hadoop systems. The results show that, compared with the traditional hybrid erasure coding techniques, EC-Fusion accelerates the response time for application by up to 1.77$\times$, and reduces the reconstruction time by up to 69.10%.

***Index Terms*—Erasure Coding, Storage Fusion, High Reliability, Reconstruction, Storage Efficiency**

I. Introduction

Nowadays, data reliability becomes a critical issue in cloud storage systems, due to the expansion of data volumes and high risks on data loss [1] [2] [3] [4] [5]. Erasure coding utilizes a small amount of redundant data as parities to ensure high fault tolerance, which is a typical technique for high reliable cloud storage. In general, erasure codes are divided into two types [6] [7], XOR-based codes [8] [9] [10] [11] [12] [13] and Reed Solomon (RS)-based codes [14] [15] [16]. Although they can maintain high fault tolerance with low storage cost, the overall performance is significantly affected by diverse factors, such as the computational and network transmission overhead during reconstruction period [2] [15] [16] [17]. As online recovery becomes frequent in cloud storage systems, it is difficult to find a perfect code to achieve high performance under both application and recovery workloads, which are foreground and background I/Os in the online recovery, respectively.

Therefore, hybrid redundancy schemes [15] [16] [18] [19] are proposed to speed up the processing of both application and recovery workloads in several directions, such as reducing I/O cost, decreasing transmission bandwidth, etc. Typical hybrid redundancy schemes can be categorized into three classes, Internal Hybrid Erasure Coding (IH-EC), External Hybrid Erasure Coding (EH-EC), and Combination of Replications and Erasure Coding (REC). IH-EC methods integrate different erasure codes into a single code via multiple dimensional constructions. Previous methods like Local Reconstruction Code (LRC) [15], Minimum Storage Regeneration (MSR) code [16] [20] [21], and Hybrid-RC [22] are internal hybrid erasure codes. General EH-EC methods include Hadoop Adaptively-Coded Distributed File System (HACFS) [18] and HeART [23], which are designed to adapt various workloads by dynamically selecting the appropriate code from a series of family codes[1]. For example, HACFS adaptively deploys erasure codes based on access patterns to optimize the recovery performance. The third class is Combination of Replications and Erasure Coding (REC) such as MIngling Chained Storage (MICS) [24] and Cocytus [25]. On the basis of various erasure codes, these methods maintain multiple copies of data (the whole data or a part of data) to enhance the reliability with competitive I/O performance.

However, existing solutions have some drawbacks on the online recovery scenarios in cloud storage systems, where the foreground application and the background recovery workloads are handled simultaneously. Regarding to IH-EC methods, the output of internal hybrid erasure coding is a static code, which cannot adapt the dynamic changing on I/O patterns or failure characteristics. Therefore, the effectiveness of these approaches is restricted.

Second, although previous EH-EC methods can reduce the reconstruction I/Os, basically they focus on single workload and still have potential to be improved via a comprehensive analysis on both application and recovery I/Os in disk arrays [26] [27] [28] [29]. Finally, REC methods result in extremely high monetary cost.

[1]The codes are from the same family [18], which can be encoded/decoded in the same way. For example, RS(4,2) and RS(6,3) can be regarded as family codes.

To address the above problems, in this paper, we propose Erasure Codes Fusion (EC-Fusion), an efficient hybrid erasure coding framework to improve the overall performance on both application and recovery workloads. Unlike HACFS and HeART using two erasure codes from the same family, EC-Fusion is a comprehensive solution to dynamically select proper codes based on the access or recovery patterns in different workloads.

The contribution of this paper include,

1) We propose Erasure Codes Fusion (EC-Fusion) to incorporate Reed Solomon (RS) and Minimum Storage Regeneration (MSR) codes, which is an effective and adaptive framework to select proper erasure codes to enhance the overall performance on both application and recovery workloads.
2) We conduct several experiments in an original hadoop system to demonstrate the effectiveness of EC-Fusion. Compared to existing hybrid erasure coding approaches, EC-Fusion is one of the best solutions to balance computation complexity and transmission overhead with a low monetary cost concurrently.

The rest of paper is organized as follows. In Section II, the related work and our motivation are introduced. In Section III, the design of EC Fusion and the related modules are illustrated in detail. The evaluation is presented in Section IV and the conclusion of our work is in Section V.

TABLE I
SYMBOLS USED IN THIS PAPER

Symbols	Description
n	total number of nodes in a storage system
r	total number of parity nodes
k	total number of data nodes ($k = n - r$)
z	the number of groups in LRC
s	an factor of n used in MSR code
m	another factor of n in MSR code ($n = sm$)
l	the integer in MSR code ($l = s^m$)
b_{ij}	the element at the ith row and jth column
d_i	the ith data element in a codeword of RS code
p_i	the ith parity element in a codeword of RS code
D_i	the ith data vector in a codeword of MSR code
P_i	the ith parity vector in a codeword of MSR code
L_i	the ith local parity node in LRC
N_i	the ith data node in LRC
G_i	the ith global parity node in LRC
H	a parity-check matrix
H_D	the submatrix correlated to data in H
H_P	the submatrix correlated to parities in H
γ	the size of a block
β	the ratio of write/read
ϕ	the number of bytes obtained by one I/O operation
W	the write overhead
R	the reconstruction cost
δ	the ratio of (writes/recoveries)
η	the threshold in EC-Fusion
α	the calculation speed (number of XOR/GF multiplications per second) in the storage system
λ	the number of bytes can be transmitted in network per second

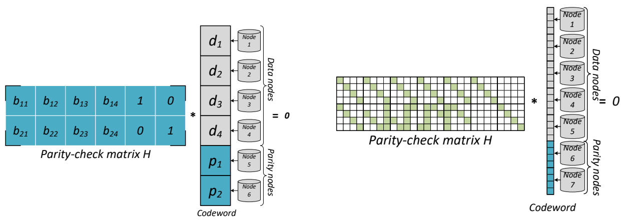
II. RELATED WORK AND OUR MOTIVATION

In this section, we briefly illustrate the hybrid redundancy schemes and related erasure codes. At the end, we discuss the motivation of this paper. To facilitate our discussion, the relevant symbols are summarized in Table I.

A. Basic Erasure Codes

The existing hybrid redundancy schemes are based on Reed Solomon (RS) based codes and XOR-based codes [6] [7]. RS-based codes are a series of variants of Reed Solomon (RS) code [14], which include Cauchy-RS [30], Local Reconstruction Code (LRC) [15], Minimum Storage Regeneration (MSR) code [16] [18]. In XOR-based codes, the parities are generated via XOR operations among data, such as EVENODD [8], X-Code [11], RDP code [9], STAR codes [31], H-Code [12], TIP-Code [13], etc. Here we take RS and EVENODD codes as examples to illustrate the generation of these two types of erasure codes.

1) Reed Solomon (RS) Code: It is a classic code based on the computations over Galois Field (GF). A specific layout of RS code can be expressed as RS(k, r), where k and r represent the number of data and parity nodes, respectively. In Fig. 1(a), we show the parity-check matrix [6] of RS code.


b_{11} b_{12} b_{13} b_{14} 1 0
b_{21} b_{22} b_{23} b_{24} 0 1
Parity-check matrix H
d_1 d_2 d_3 d_4 p_1 p_2
Node 1 Node 2 Node 3 Node 4 Node 5 Node 6 Node 7
Data nodes
Parity nodes
Codeword
= 0

(a) The construction of RS code. (b) The construction of EVENODD.

Fig. 1. The construction of RS and EVENODD codes. (For RS code, a square stands for an element in Galois Field; For EVENODD, a square denotes a bit, and a white square represents bit 0 while a green square denotes bit 0.)

2) EVENODD Code: As shown in Fig. 1(b), EVENODD is composed of horizontal and diagonal parities, which are generated by XOR operations among data and can tolerate two concurrent failures.

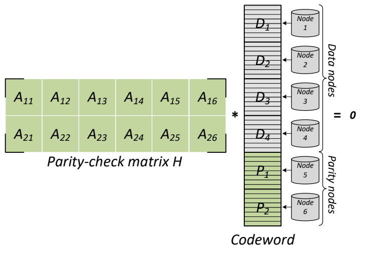
According to the above, single codes can be easily constructed, which provide high efficiency for application I/Os. However, a large amount of the reconstruction bandwidth causes high latency on recovery workloads.

B. Internal Hybrid Erasure Coding (IH-EC)

Internal hybrid erasure coding (IH-EC) combines two different erasure codes via multiple dimensional constructions. Typical IH-EC methods involve GRID Codes [32], H-Code [12], HV Code [33], RDP Code [9], HDP Code [10], X-Code [11], Code 5-6 [34], MSR code [16] [18], LRC [15], Hybrid-RC [22], AZ-Code [35], etc. Here we give a brief introduction to MSR and LRC codes.

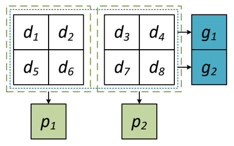
1) Minimum Storage Regeneration (MSR) Code: The construction of MSR code is based on RS code and enhances the coupling among data. As shown in Fig. 2(a), an MSR code can be represented as MSR(n, k, r, l) $(n = sm,\ l = s^m)$, where n, k, r and l are the number of total nodes, data nodes, parity nodes, and the dimensions of vectors, respectively.

2) Local Reconstruction Code (LRC): As shown in Fig. 2(b), LRC is a mixture of RS and XOR-based codes, which improves the recovery performance with flexible lengths of parity chains. An LRC code is denoted as LRC(k, r, z), where k, r, z are the number of data nodes, global parity nodes, local parity nodes, respectively.



A_{11} A_{12} A_{13} A_{14} A_{15} A_{16}
A_{21} A_{22} A_{23} A_{24} A_{25} A_{26}
Parity-check matrix H
D_1 D_2 D_3 D_4 P_1 P_2
Node 1 Node 2 Node 3 Node 4 Node 5 Node 6
Data nodes
Parity nodes
= 0
Codeword

(a) The construction of MSR Code.



(b) The construction of LRC.

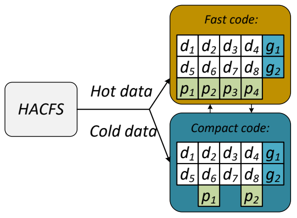
Fig. 2. The construction of LRC and MSR codes. (For MSR code, Parity-check matrix of MSR(n, k, r, l) contains $r \times n$ sub-matrices A_{ij} with the size of $l \times l$, where $i \in [1, r]$ and $j \in [1, n]$. D and P are l-dimensional element vectors, and all of them form a codeword in MSR code; For LRC, global parities g are generated from RS code while local parities p are calculated by XOR operation among local data. Here $p_1 = d_1 \bigoplus d_2 \bigoplus d_5 \bigoplus d_6$, and $p_2 = d_3 \bigoplus d_4 \bigoplus d_7 \bigoplus d_8$.)

Although IH-EC methods can significantly improve the recovery performance, the lack of adaptation to I/Os reduces the effects on the overall performance.

C. External Hybrid Erasure Coding (EH-EC)

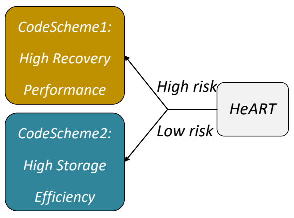
Different from IH-EC methods, external hybrid erasure coding (EH-EC) incorporates a series of family codes and dynamically selects the appropriate one to adapt application or recovery workloads. In this part, we mainly describe two EH-EC schemes, HACFS [18] and HeART [23].

1) HACFS: Hadoop Adaptively-Coded Distributed File System (HACFS) [18] uses two erasure codes (namely fast code and compact code) from the same XOR-based family codes. Fast code provides high recovery performance for hot data, and compact code aims to save the storage cost for cold data, which are shown in Fig. 3(a).



HACFS
Hot data
Cold data
Fast code:
d_1 d_2 d_3 d_4 g_1
d_5 d_6 d_7 d_8 g_2
p_1 p_2 p_3 p_4
Compact code:
d_1 d_2 d_3 d_4 g_1
d_5 d_6 d_7 d_8 g_2
p_1 p_2

(a) HACFS.



CodeScheme1: High Recovery Performance
CodeScheme2: High Storage Efficiency
High risk
Low risk
HeART

(b) HeART.

Fig. 3. The basic framework of HACFS and HeART methods. (HACFS: hot data in applications→fast code, cold data in applications→compact code. HeART: high risk on data loss → code scheme 1, low risk on data loss→code scheme 2.)

2) HeART: HeART [23] utilizes heterogeneous levels of reliability to tradeoff the storage and recovery performance in a long term. The basic idea of HeART is to adopt a code with high recovery speed for high risk data, and another code for high storage efficiency is applied during low risk period, as shown in Fig. 3(b).

Based on the above, HACFS further decreases the recovery overhead by adaptation to application I/Os, and HeART effectively reduces monetary cost with high reliability via analysis on failure characteristics. However, the one-sided adaptation of the existing EH-EC methods cannot maximize thier effectiveness.

D. Combination of Replications and Erasure Coding (REC)

Replications and Erasure Coding (REC) are highly complementary on recovery performance and monetary cost, so they are combined together for comprehensive solutions such as MICS [24] , Cocytus [25], EC-Cache [36], Diskreduce [37], Encoding-Aware Replication [38], etc. Here we give a brief introduction to MICS and Cocytus methods.

1) MICS: MIngling Chained Storage (MICS) [24] is a general method to incorporate replications and erasure codes. Fig. 4 shows the construction of MICS, which provides both high efficiency on both computation and reconstruction.

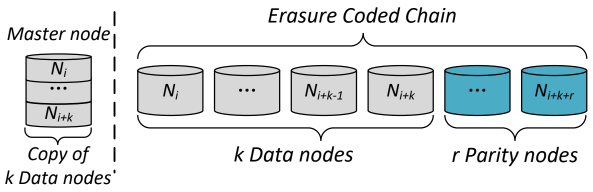


Fig. 4. The basic framework of MICS. (MICS: MICS maintains a full copy of data object in MN, and in ECC, k segments of the object are encoded by RS code to generate r parity segments.)

2) Cocytus: Cocytus [25] is designed to achieve high efficiency on both storage and encoding/decoding for in-memory KV-store. Generally, Cocytus dynamically uses replications for small-sized and scattered data while applying erasure code for relatively large data.

Replications can bring the improvement on the performance but lead to high storage cost, which means that REC approaches are not appropriate for large-scale systems.

E. Our Motivation

We summarize the properties of various hybrid redundancy approaches in Table II. It is clear that IH-EC methods cannot adapt the dynamic change on both application and recovery workloads. When a sharp alteration on application or recovery data stream, the overall performance are restrictedly limited.Due to the extremely high storage cost, REC methods are not suitable for large-scale distributed storage systems, either. For EH-EC methods, the usage of the same family codes and one-sided analysis on application or recovery workload limit the potential to improve the overall performance for online recovery.

From Table II, an EH-EC method with adaptive adjustment on both application and recovery workloads are highly desired. Therefore, from the existing erasure codes, selecting

TABLE II
SUMMARY OF HYBRID REDUNDANCY SCHEMES

Hybrid Redundancy Methods	Name	Storage Cost	Performance on Application Workloads				Performance on Recovery Workloads			
			Computation	Transmission	I/O	Adaptive?	Computation	Transmission	I/O	Adaptive?
Internal Hybrid ErasureCoding (IH-EC)	LRC code	high	high	low	high	×	high	high	high	×
	MSR code	low	low	high	high	×	low	high	high	×
	AZ-Code	high	low	low	high	×	low	high	high	×
	Hybrid-RC	high	low	low	high	×	low	high	high	×
External Hybrid Erasure Coding (EH-EC)	HACFS	medium	high	low	high	√	high	high	high	×
	HeART	low	high	high	high	×	high	low	high	√
	EC-Fusion	**low**	**high**	**high**	**high**	√	**high**	**high**	**high**	√
Combination of Replications & Erasure Coding (REC)	MICS	very high	low	medium	medium	×	low	low	low	×
	Cocytus	very high	low	N/A	N/A	×	low	N/A	N/A	×

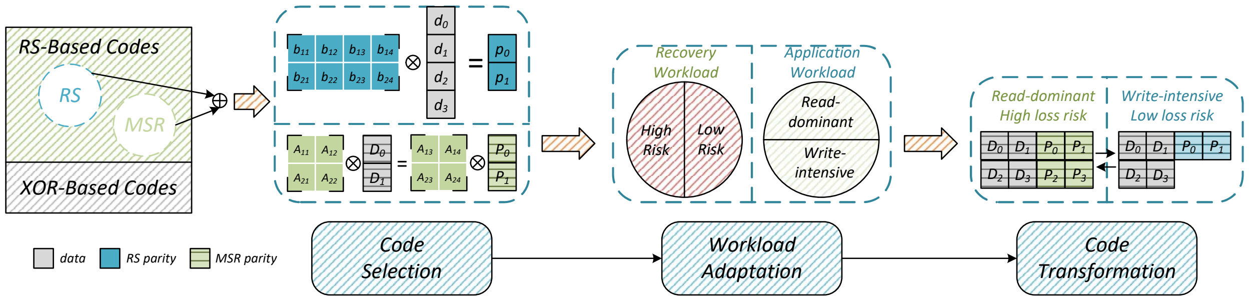


Fig. 5. The framework of EC-Fusion.

appropriate codes, analyzing both application and recovery I/Os, and then designing an adaptive hybrid coding scheme via externally combining the codes can be a proper way to achieve complementarity and maximize the overall performance, which motivates us to propose a novel EC-Fusion in this paper.

III. EC-FUSION

In this section, the design of EC-Fusion is introduced in detail. Compared to the existing EH-EC methods, the key idea of EC-Fusion is to maximize the overall performance based on a global point view on both application and recovery workloads. The following sections describe an overview of EC-Fusion and the corresponding main modules in detail.

A. Overview of EC-Fusion

To achieve the above design purpose, EC-Fusion mainly includes three modules, *Code Selection*, *Workload Adaptation*, and *Code Transformation*, which are shown in Fig. 5. Here is a brief illustration of these modules as below,

1) *Code Selection*: Giving a brief understanding to the encoding/decoding of hybrid erasure codes, and then selecting the appropriate parameters for them.
2) *Workload Adaptation*: Statically allocating a proper coding scheme for each of them, and providing adaptive rules to dynamically adjust the hybrid coding schemes.
3) *Code Transformation*: Providing a fusion approach to achieve efficient conversions among different codes.

A combination of XOR-based codes is introduced in HACFS [18], so in this paper we mainly focuses on the fusion of RS-based codes and introduces each module in detail.

B. Code Selection

In order to select a pair of appropriate codes to adaptively adjust both application and recovery workloads, we summarize the following requirements,

1) the selected codes should have similar layout, which means that it is possible to establish the conversions among them.
2) the selected codes should have high flexibility to support arbitrary number of storage nodes, which are suitable for dynamic application and recovery workloads.
3) the selected codes play different roles in application and recovery workloads, respectively. They can give us an opportunity to externally combine them and achieve complementarity.

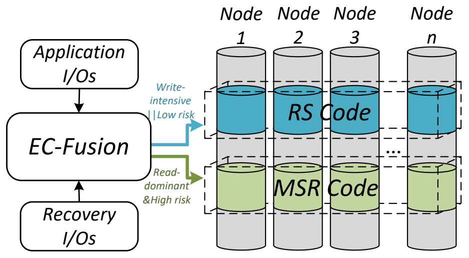
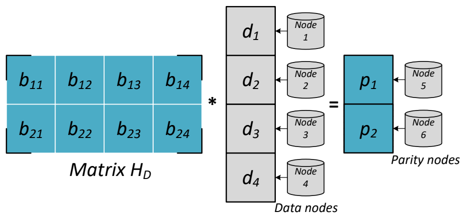
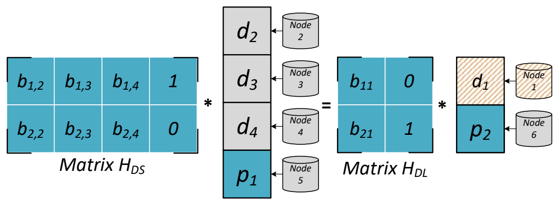


Fig. 6. A sample design of EC Fusion. (EC-Fusion adopts RS code for write-intensive data in application workloads or low probabilities on data loss, and selects MSR code for read-intensive data in application workloads and high probabilities on data loss.)

Except for XOR-based codes in HACFS [18], RS-based codes can be also be candidates. Firstly, we select RS and MSR codes as an EH-EC method (shown in Fig. 6). As we know, both RS and MSR codes can be calculated over Galois Field. RS and MSR codes have high scalability for most of storage systems. Moreover, on one hand, as shown in Fig. 7(a) and 8(a), since the construction of the matrices involved in MSR code is complex, RS code has low computational cost, which provides lower overhead for write requests in applications. On the other hand, according to Fig. 7(b) and 8(b), the total amount of data and parities for single disk failure reconstruction is decreased in the MSR decoding process. Therefore, MSR has great advantage on reducing the network transmission cost for recovery, which can help to accelerate the processing of recovery workloads.

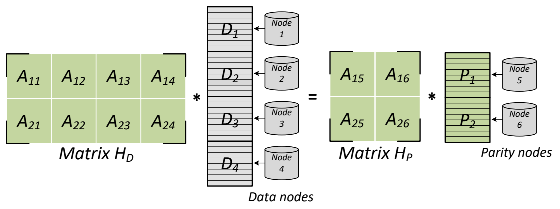


(a) Encoding of RS code

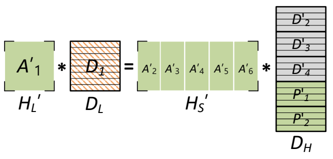


(b) Decoding of RS code

Fig. 7. The encoding/decoding processes of RS Code. (In the encoding process, r parity elements are generated via GF calculations over k data elements. In the decoding process, the lost data element is recovered via the any k survivors of data or parities, which means that extra network transmission is needed during the recovery process.)



(a) Encoding of MSR code


A'_1
H_L'
D_1
D_L
A'_2 A'_3 A'_4 A'_5 A'_6
H_S'
D'_2 D'_3 D'_4 P'_1 P'_2
D_H

(b) Decoding of MSR code

Fig. 8. The encoding/decoding processes of MSR Code. (In the encoding process, r parity vectors are generated from the multiplication of the $rl \times nl$ matrix H_D and k data vectors. In the decoding process, MSR code transforms the matrix H_D and H_P to H'_L and H'_S, and fetches $(1/r)$ of each remaining data and parity vectors to restore the lost vector D_L.)

Next, we choose appropriate parameters for RS and MSR codes, based on the encoding/decoding processes of RS(k, r) and MSR(n, k, r, l). The application and recovery performance on a single block are summarized in Table III.

TABLE III
PERFORMANCE [2] COMPARISONS AMONG HYBRID CODE SCHEMES

Code Scheme		RS(k, r)	MSR(n, k, r, l)
Application Performace	Computational Cost	$\frac{\beta}{1+\beta}\gamma kr$	$\frac{\beta}{1+\beta}(l^3 + l \times \gamma kr)$
	Disk I/O Cost	γ/ϕ	
	Transmission Cost	$\frac{\beta(r+k)/k+1}{1+\beta}$	
Recovery Performance	Computational Cost	$nr^2 + \gamma k$	$l^3 + l \times \gamma \frac{n-1}{r}$
	Disk I/O Cost	γ/ϕ	$\gamma/(r\phi) \sim \gamma/\phi$
	Transmission Cost	k	$(n-1)/r$

Finally, EC-Fusion selects RS(k, r) and MSR$(2r, r, r, r^2)$ as the hybrid coding strategy, namely EC-Fusion(k, r). In this way, we can obtain the following benefits,

1) RS(k, r) guarantees high storage efficiency for the whole system, and keeps high application performance for write-intensive I/Os.
2) MSR$(2r, r, r, r^2)$ provides high recovery performance for high risk data and incurs low side-effect for read-dominant I/Os.
3) The conversion between MSR$(2r, r, r, r^2)$ and RS(k, r) can be conveniently added to the original coding scheme.

C. Workload Adaptation

Based on the characteristics of I/Os, we statically divide them into six categories (shown in Table IV), 1) cold data with low risk; 1) cold data with high risk; 3) write-intensive data with low risk; 4) write-intensive data with high risk; 5) read-dominant data with low risk; 6) read-dominant data with high risk. Here both read-dominant and write-intensive data belongs to hot data, and the application with balanced read and write can belong to write-intensive or read-dominant. In general, under a low risk environment, this application is more likely to be write-intensive, since the overall performance is mainly affected by write operations. Otherwise, it can be treated as read-dominant. Then we determine which code to allocate for them, and provide adaptive rules for dynamic I/Os.

1) Code Allocation: Here we allocates appropriate coding schemes to workloads with matching features. And we summarize the code allocation in Table IV.

Intuitively, it is no doubt that cold data or low risk data are encoded with RS code, which keeps high storage efficiency and guarantees good application performance, especially for write-intensive I/Os. For read-dominant data with high failure risk, MSR code can save a lot of reconstruction bandwidth and cause few side effects to application I/Os.

However, it seems to be difficult to choose a proper code scheme for write-intensive data with high risk, since we should

[2] In this table, we evaluate the performance of the traditional RS code and a specific MSR code [20]. And the computational cost are approximate.

TABLE IV
CODE ALLOCATION FOR VARIOUS WORKLOADS IN EC-FUSION

Code Scheme		Recovery Workload	
		High Risk	Low Risk
Application Workload	Write Intensive	MSR or RS	RS
	Read Dominant	MSR	RS
	Cold	RS	RS

trade off application and recovery I/Os and determine a proper code to maximize the overall performance. To solve this problem, we define the variables (based on Table III) as below,

$$W_{RS} = \gamma(kr/\alpha + ((k+r)/k)/\lambda + 1/\phi),$$
$$R_{RS} = (nr^2 + \gamma k)/\alpha + \gamma(k/\lambda + 1/\phi),$$
$$W_{MSR} = r^4(r^2+\gamma)/\alpha + \gamma(2/\lambda + 1/\phi),$$
$$R_{MSR} = (r^6 + \gamma(2r^2 - r))/\alpha + \gamma((2r-1)/(r\lambda) + 1/\phi).$$

Then we obtain the following inequality to determine which code scheme to apply,

$$\delta = \frac{writes}{recoveries} \geq \frac{R_{RS} - R_{MSR}}{W_{MSR} - W_{RS}} = \eta. \quad (1)$$

And $\delta \geq \eta$ means that the overall performance is mainly affected by write requests, so RS code is preferred; otherwise, the reconstruction bandwidth is the main factor, and then MSR code becomes a better choice. In practice, in order to reduce the overhead caused by frequently switching the schemes, we can adjust the inequality (1) as following,

$$\delta \geq \eta + \Delta, or\ \delta \leq \eta - \Delta,\ (0 \leq \Delta < \eta). \quad (2)$$

2) Adaptive Rules: Since both data accesses and failures usually exhibit temporal and spatial localities [39] [40] [41] [42] [43], EC-Fusion can apply existing cache algorithms (such as LRU, LFU etc), for the identification of hot or cold data. In EC-Fusion, two queues are used for access patterns and failure characteristics, respectively. And existing cache algorithms can be applied in these queues.

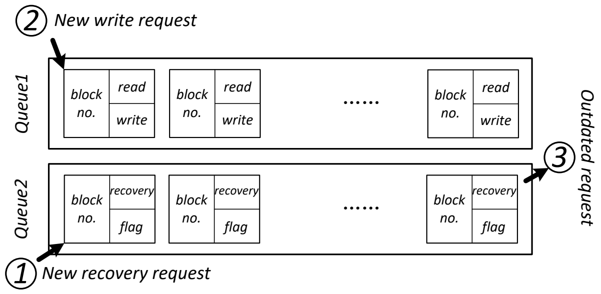


Fig. 9. Two queues are used for capturing the data access patterns and failure characteristics, respectively. (Three conditions to trigger an adaptive process are marked with ①, ②, and ③.)

As shown in Fig. 9, we use a queue named ”Queue1” to log the information of data accesses, which records the block IDs and the number of cache hits. ”Queue2” stores the recovery requests to each block, including the block IDs, the number of cache hits, and flags indicating coding scheme.

Initially, we set RS(k,r) as the default code for the whole storage system, and these queues can have a warm-up after the startup of the system. In adaptive rules, there are three conditions needed to trigger an adaptive process, which is shown in Fig. 9. And the adaptive processes are based on the inequality (1). First, for each recovery request being inserted at the head of Queue2, EC-Fusion should determine the related blocks whether they need to be allocated with MSR code. Second, for each write request being inserted at the head of Queue1, EC-Fusion should determine the related block whether need to be encoded by RS code. Third, for each recovery request being deleted at the tail of Queue2, EC-Fusion should convert the MSR code to RS code. And we summarize the adaptive rules in Algorithm 1.

Algorithm 1 Adaptive Selection Algorithm in EC-Fusion

```
Input: Queue1, Queue2 and $\eta$;
for each request inserted at the head of Queue2 do
  if flag $\neq$ MSR and (writes/recoveries) $<\eta$ then
    set flag = MSR;
    convert to MSR code;
  end if
end for
for each request inserted at the head of Queue1 do
  if flag $\neq$ RS and (writes/recoveries) $\geq \eta$ then
    set flag = RS;
    convert to RS code;
  end if
end for
for each request deleted at the tail of Queue2 do
  if flag == MSR then
    set flag = RS;
    convert back to RS code;
  end if
end for
```

D. Code Transformation

In order to cope with dynamical workloads, code transformation between the hybrid erasure codes is necessary, which is the core module in EC-Fusion.

Based on the construction of RS(k,r), we can separate the parity-check matrix into several submatrices with the size of $r \times r$, as shown in Fig. 10. When r cannot divide k, empty data nodes are added. For example, one empty data node can be added into RS(8,3). For convenience, assume that k can be divisible by r and $k = qr$. And then the equation of parity generation can be rewritten as below,

$$\mathbf{p} = \sum_{i=1}^{q} B_i \times \mathbf{d}_i = \mathbf{p}'_1 + \mathbf{p}'_2 + \cdots + \mathbf{p}'_q, \quad (3)$$

where $\mathbf{p} = \{p_1, p_2, \cdots, p_r\}^T$, B_i is a $r \times r$ submatrix, $\mathbf{d}_i = \{d_{(i-1)r+1}, \cdots, d_{(i-1)r+r}\}^T$, $\mathbf{p}'$ stands for the intermediary parity. In this way, $\mathbf{p}$ can be viewed as the combination of $(\mathbf{p}'_1, \mathbf{p}'_2, \cdots, \mathbf{p}'_q)$.

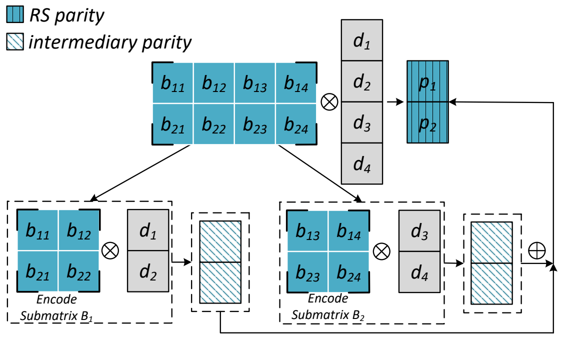


Fig. 10. The separation of the construction in RS(k, r). (In this process, the encoding process of RS(k, r) is divided into (k/r) sub-encoding processes, which generate the (k/r) sets of intermediary parities.)

Since any $r \times r$ submatrix of parity-check matrix in RS code is invertible, each intermediary $\mathbf{p}'_i$ has the following property,

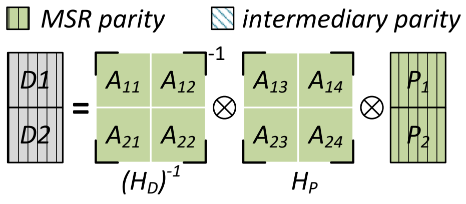
$$\mathbf{d}_i = B_i^{-1} \times \mathbf{p}'_i. \tag{4}$$

As we know, MSR$(2r, r, r, r^2)$ has the same property shown in Fig. 11(a). In order to match the scale of RS and MSR codes, we extend the element d_j in $\mathbf{d}_i$ to the vector D_j as "$d_j \to D_j = \{d_{j1}, d_{j2}, \ldots, d_{jl}\}^T$", where $l = r^2$. And then we obtains "$\mathbf{D}_i = \{D_{(i-1)r+1}, \cdots, D_{(i-1)r+r}\}^T$".

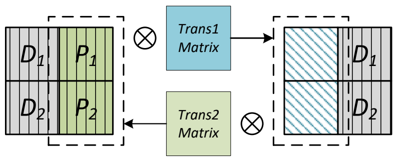
The related intermediary parity $\mathbf{p}'_i$ is extended to $\mathbf{P}'_i$. For the inverted matrix B_i^{-1}, each element b' in B_i^{-1} is extended to a $l \times l$ diagonal matrix $diag(b', b', \ldots, b')$. Thus, an extension of B_i^{-1} contains $r \times r$ submatrices with the size of $l \times l$, which is denoted as $\mathbf{B}'_i$. Then the equation (4) can be rewritten by,

$$\mathbf{D}_i = \mathbf{B}'_i \times \mathbf{P}'_i. \tag{5}$$

Similarly, the original matrix B_i can be extended as $\mathbf{B}_i$.


MSR parity
intermediary parity
D1 D2
A_{11} A_{12} A_{21} A_{22}
-1
A_{13} A_{14} A_{23} A_{24}
P_1 P_2
$(H_D)^{-1}$
H_P

(a) MSR$(2r, r, r, r^2)$.



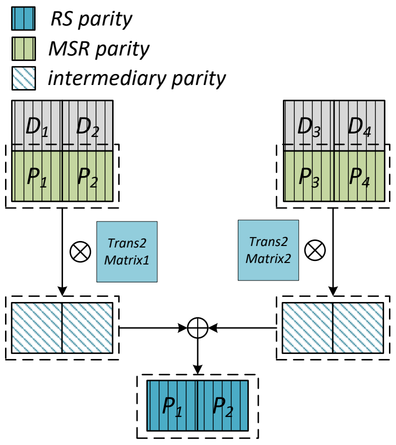
(b) The conversion between the intermediary and MSR parities.

Fig. 11. The establishment of the transformation between the intermediary and MSR parities.

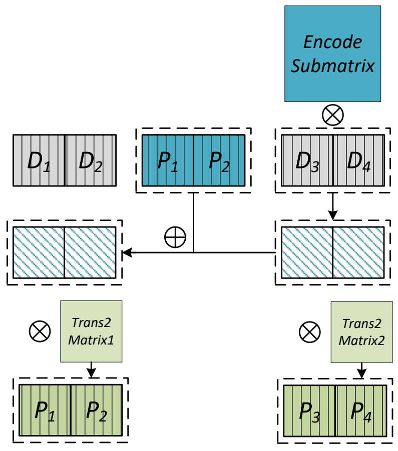
Then we can easily establish the conversion between the intermediary parities and the parities of MSR$(2r, r, r, r^2)$, as shown in Fig.11(b), where "Trans1" and "Trans2" can be derived as following,

$$Trans1 = \mathbf{B}_i \times H_D^{-1} \times H_P, \tag{6}$$
$$Trans2 = H_P^{-1} \times H_D \times \mathbf{B}'_i. \tag{7}$$

Therefore, the intermediary parities can be served as a highway to achieve efficient transformation between RS and MSR. As shown in Fig. 12(a), for the transformation from MSR code to RS code, (k/r) sets of the original parities can be transformed into the intermediary parities, and then they are merged into the final RS parities.



(a) MSR to RS.



(b) RS to MSR.

Fig. 12. Transformation between MSR and RS codes.

The conversion from RS to MSR can be viewed as a reverse process, where one set of the original RS is divided into the final (k/r) sets of MSR. The detail is shown in Fig. 12(b), where $(k/r) - 1$ sets of data are multiplied by their corresponding submatrices. In this way, the intermediary parities of the last set can be obtained by all existing parities, and then all the intermediary parities are transformed into the final MSR code.

IV. Evaluation

In this section, we conduct a series of mathematical analysis and experiments to demonstrate the effectiveness of EC-Fusion under various application and recovery workloads.

A. Evaluation Methodology

To evaluate the effectiveness of EC-Fusion, we select the following hybrid erasure coding methods in our evaluation,

- *IH-EC methods*: MSR code [16] [20], LRC [15].
- *EH-EC methods*: HACFS [18] method.

And we add RS code in our comparisons as well. Because AZ-Code [35] and Hybrid-RC [22] adopt an idea similar to LRC, HeART method is implemented for long term recovery optimization, and REC methods have extremely high storage cost, so they are excluded in our comparisons.

TABLE V
Summary of traces

Trace	# of Requests	Read%	IOPS	Avg. Req. Size
MSR-mds1	1637711	92.88%	27.29	113.00 KB
MSR-rsrch2	207597	65.69%	3.54	8.17 KB
MSR-web1	160891	54.11%	2.66	58.14 KB
MSR-rsrch0	1433655	9.32%	23.70	17.86 KB

1) Application Workloads: We select four traces [44] as the typical application workloads shown in Table V. Here we give a brief introduction to them as following.

- *MSR-mds1* is traced from media server with the highest read percentage among the selected traces.
- *MSR-rsrch0* and *MSR-rsrch2* are generated by research projects. *rsrch0* has the lowest percentage while *rsrch2* gets medium read percentage.
- *MSR-web1* is traced from Web/SQL server with the medium read percentage.

2) Recovery Workloads: We initialize the generation of failures randomly, and then records the locations and timestamps of the failures. For temporal locality, we calculate the time intervals from the last time when failures occur, and generate failure probabilities by normal distribution with the interval inputted. Regarding to spatial locality, the relative distances between the target location and the nearest failure is inversely proportional to the failure probabilities. Since 98% failures in storage systems are single disk failures [18], we evaluate the recovery performance of single chunk failures in our experiments.

3) Metrics and Methods for Mathematical Analysis: We use the **Storage Cost**, **Computational Cost**, and **Transmission Cost** as the metrics for mathematical analysis.

(1.a) *Storage Cost* is defined as $\rho = (k+r)/k$ in a storage system with k data nodes and r parity nodes.

(1.b) *Computational Cost* is defined as the number of GF-multiplication/Exclusive-OR operations in the application and recovery workloads.

(1.c) *Transmission Cost* is defined as the number of chunks transmitted on the application and recovery workloads.

As EH-EC methods adjust the parity chains to adapt workloads, the above metrics can also change dynamically. To simplify the evaluation, we select various combination ratios of two erasure codes in EH-EC schemes.

4) Metrics and Methods for Experiments: We use the **Application Performance**, **Recovery Performance**, **Overall Performance**, and **Cost-effective Ratio** as the metrics. They are defined as following,

(2.a) *Application Performance* ϵ_1 is measured by the average latency of read and write operations on application workloads, which contains computational and data access (transmission + disk I/Os) costs.

(2.b) *Recovery Performance* ϵ_2 is measured by the average overhead of decoding on recovery workloads, including computational and access costs.

(2.c) *Overall Performance* ϵ is defined by, $\epsilon = (\mu_1\epsilon_1 + \mu_2\epsilon_2)/(\mu_1 + \mu_2)$, where μ_1 and μ_2 represents the number of requests on application and recovery workloads, respectively.

(2.d) *Cost-effective Ratio* ζ is defined as $\zeta = 1/(\epsilon \times \rho)$, which is the ratio between the whole performance and the storage cost.

5) Experimental Environment: The environment of our experiments is shown in Table VI. We set up a Hadoop system to evaluate the performance of hybrid erasure coding schemes. And requests to files in HDFS are used for the simulation of application workloads, and each file is composed of k chunks. In order to facilitate the implementation of MSR, each chunk is set to 27 MB. Since a file can only be written once in HDFS, we treat each write request in traces as a new write, and each read request reads one data chunk. Finally, we implement EC-Fusion and the mentioned coding schemes, and use a test program to obtain their application/recovery performances during workloads.

TABLE VI
Parameters of our evaluation platform

Description	DELL R730 Server
CPU	Intel Xeon 3.0GHz
NIC	1Gbps
Memory	32GB
Disk	3TB SSD
OS	Ubuntu 16.04
Platform	Hadoop HDFS 3.1.2

Our evaluation consists of five parts: 1) mathematical analysis on hybrid erasure coding schemes, 2) experiments to evaluate the application performance, 3) experiments to evaluate the recovery performance, 4) evaluation on the whole performance, 5) evaluation on the cost-effective ratio.

B. Numerical Results of Mathematical Analysis

In this section, we show the numerical results of mathematical analysis on the storage cost, computational cost, and transmission cost for EC-Fusion and other hybrid erasure coding schemes. Here, we set RS$(k, 3)$ $(k = 6,\ 8)$ as the baseline, and list the mentioned coding schemes as following,

1) Internal Hybrid Erasure Coding Schemes:

- MSR code: MSR$(k+3, k, 3, l)$. when $k = 8$, one virtual nodes are added into the systems.

- LRC: LRC$(k, 2, 2)$ with two additional local parities and the same fault tolerance with RS$(k, 3)$.

2) External Hybrid Erasure Coding Schemes:

- EC-Fusion: EC-Fusion$(k, 3)$ is a combination of RS$(k, 3)$ and MSR$(6, 3, 3, 9)$.
- HACFS: HACFS-k is a combination of LRC$(k, 2, 2)$ and LRC$(k, 2, k/2)$.

1) Storage Cost: In this part, we compare the storage cost of the above approaches. Typically, RS code and the IH-EC methods have a constant value for storage cost, while EH-EC methods adopt different coding schemes for application and recovery workloads. Thus, we set a hybrid ratio $h\%$ for MSR applied in EC-Fusion and fast code (ie, LRC$(k, 2, k/2)$) used in HACFS. The result is shown in Fig.13, EC-Fusion increases 9.1% at most ($k = 8$) to the original RS, but never exceeds LRC or HACFS.

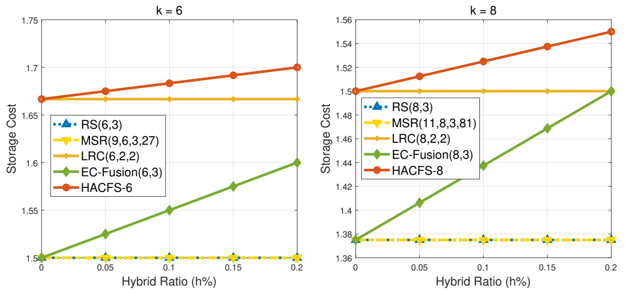


Fig. 13. Mathematical analysis on storage cost.

2) Computational Cost: In this part, we shows the computational cost (defined in Section IV-A3) on application and recovery workloads, respectively. Fig.14 shows the results for the scenarios with $k \times 64$ KB data (one stripe) written and 64 KB data (one column in a stripe) reconstructed. As shown in Fig.14, EC-Fusion basically keeps the computational efficiency with RS, LRC and HACFS. Compared to MSR, EC-Fusion can saves at least 96.30% and 79.24% computational cost for application and recovery workloads, respectively.

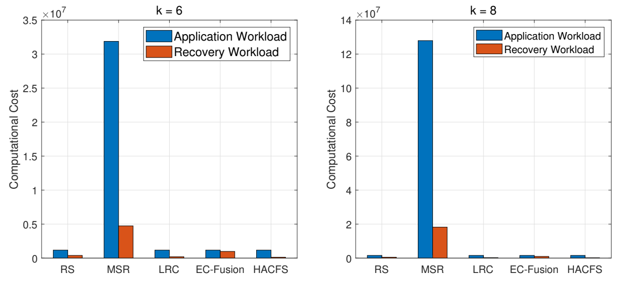


Fig. 14. Mathematical analysis on computational cost.

3) Transmission Cost: In this section, we discuss the transmission cost of various coding schemes on application and recovery workloads separately. Fig.15(a) shows the results for application workloads, where one stripe with k data chunks is written. Compared to LRC and HACFS, EC-Fusion can save the transmission cost at least 8.33%.

For recovery workload, we assume EH-EC methods can improve all recovery requests. As the results are shown Fig.15(b), EC-Fusion reduces the transmission up to 79.12% compared to the original RS, and improves the transmission performance at least by 16.67% compared with HACFS.

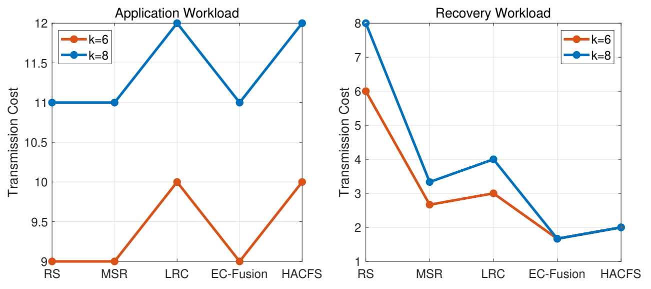


Fig. 15. Mathematical analysis on transmission cost.

C. Results of Application Performance

According to Fig.16, EC-Fusion causes no more than 1.04% overhead to the original RS. Besides, EC-Fusion improves the application performance up to 78.03% for MSR, and 10.81% for LRC and HACFS.

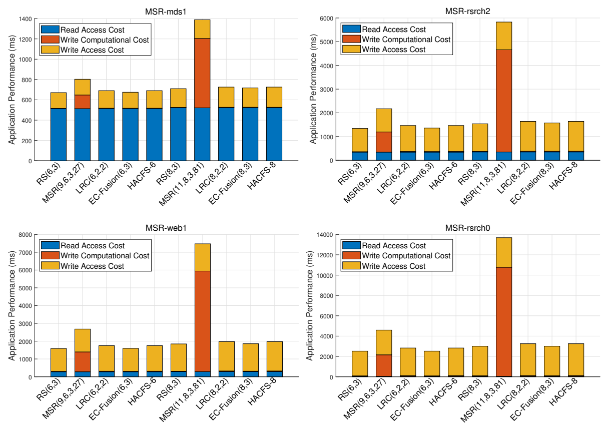


Fig. 16. Application performance under different application workloads.

D. Results of Recovery Performance

As the results shown in Fig.17, compared with RS, MSR and LRC, EC-Fusion saves up to 67.83%, 69.10% and 38.36% of the recovery overhead, respectively. Since EC-Fusion should balance both application and recovery I/Os, HACFS performs better in terms of recovery performance.

E. Results of Overall Performance

As shown in Fig.18, EC-Fusion always gains better overall performance than MSR and LRC by up to 77.98% and

TABLE VII
Improvement of EC-Fusion over other erasure coding schemes in terms of the overall performance and cost-effective ratio.

Code	k	r	Overall Performance				Cost-effective Ratio			
			MSR-mds1	*MSR-rsrch2*	*MSR-web1*	*MSR-rsrch0*	*MSR-mds1*	*MSR-rsrch2*	*MSR-web1*	*MSR-rsrch0*
RS	8	3	18.15%	4.24%	2.08%	0.60%	16.71%	2.71%	1.40%	0.23%
	6	3	14.28%	3.84%	1.86%	0.48%	13.17%	2.69%	1.36%	0.22%
MSR	8	3	51.45%	72.55%	74.51%	77.98%	50.59%	72.11%	74.33%	77.90%
	6	3	19.59%	37.08%	39.56%	45.00%	18.55%	36.34%	39.25%	44.85%
LRC	8	2+2	6.37%	4.75%	5.18%	7.61%	12.66%	11.29%	12.48%	15.00%
	6	2+2	5.20%	7.32%	7.96%	10.81%	13.58%	15.60%	16.74%	19.52%
HACFS	8	4(6)	0.16%	3.75%	6.63%	7.48%	19.72%	22.80%	25.80%	26.93%
	6	4(5)	1.72%	7.09%	9.79%	10.75%	10.76%	15.67%	18.65%	19.56%

10.81%. Compared with RS, EC-Fusion gets 18.15% improvement on overall performance in the read-dominant environment, and in the write-intensive workload, EC-Fusion achieves a 10.75% performance improvement over HACFS. Besides, the extra cost for EC-Fusion is included in the overall performance, and accounts for up to 1.47% of the overall performance.

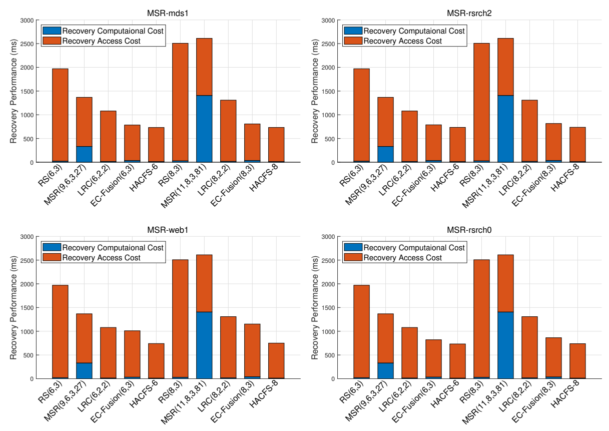


Fig. 17. Recovery performance under online recovery workloads.

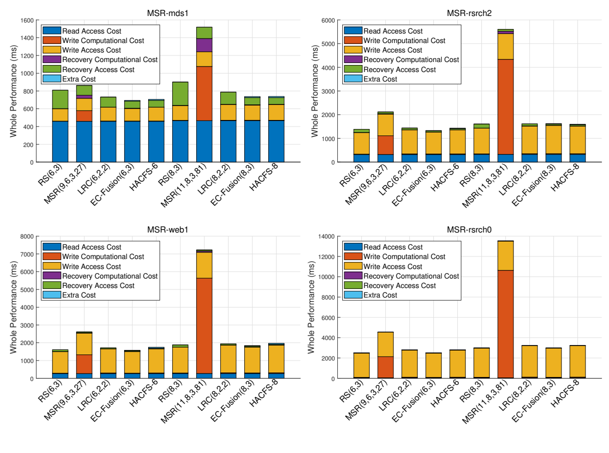


Fig. 18. Overall performance under different application and online recovery workloads.

F. Results of Cost-effective Ratio

The results of the cost-effective ratio are shown in Fig. 19. Compared with RS and MSR, EC-Fusion obtains better cost-effective ratio by up to 16.71% and 77.90%, respectively. Compared to LRC and HACFS, EC-Fusion increases the ratio by up to 19.52% and 26.93%, respectively.

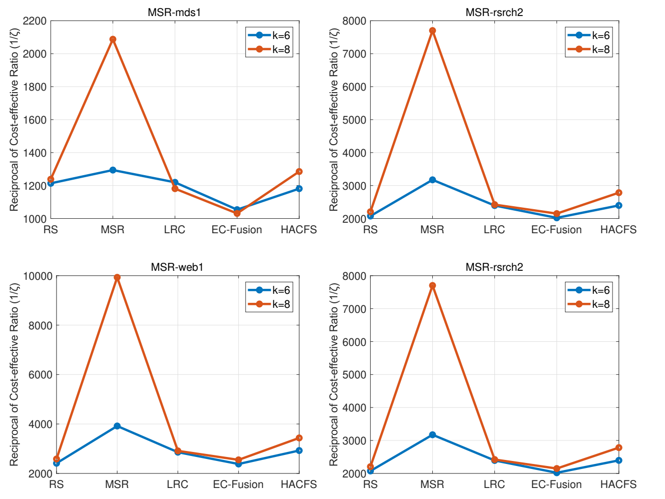


Fig. 19. Cost-effective ratio under different application and online recovery workloads.

G. Analysis

We summarize the results in Table VII. From these results listed in this section, it is clear that EC-Fusion have great advantages compared to other hybrid erasure coding approaches. There are several reasons to achieve these gains. First, EC-Fusion is a comprehensive solution based on RS and MSR codes, which are popular codes to provide high efficiency on fast decoding. Second, EC-Fusion selects proper codes according to a global point of view on access/failure patterns, which dynamically improves the effects on storage systems. Third, RS and MSR codes provide flexible configurations (i.e., supports arbitrary number of data nodes), which are easily adapted to varying fusion rules. Fourthly, the transformation overhead between RS and MSR code is very low (shown in Fig. 18), which guarantees the overall efficiency of EC-Fusion.

V. Conclusion

In this paper, we propose EC-Fusion, a hybrid erasure coding method to integrate RS and MSR codes together for the balance of application and recovery I/Os. For write-intensive workloads with infrequent decoding, we select RS code to decrease the computational overhead and storage cost. On the other hand, when read-dominant workloads with frequent decoding, MSR code is a proper choice. Therefore, the storage and decoding costs can be balanced. To demonstrate the effectiveness of EC-Fusion, we conduct several experiments. Compared to the typical hybrid erasure coding methods, EC-Fusion 1) improves up to 78.03% by reducing computational cost on write-intensive I/Os in application workloads compared to MSR code; 2) decreases the recovery latency by up to 67.83% compared to the original RS; 3) improves whole performance up to 10.75% compared to HACFS and 4) gains higher cost-effective ratio up to 19.52% compared to LRC.

VI. Acknowledgments

We thank anonymous reviewers for their insightful comments. This work is partially sponsored by the National Key R&D Program of China (No.2018YFB0105203), the Natural Science Foundation of China (NSFC) (No.61972246), the Natural Science Foundation of Shanghai (No.18ZR1418500), the Alibaba Group through Alibaba Innovative Research (AIR) program, and the U.S. National Science Foundation grant CCF-1813081. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies.

References

[1] L. Qian *et al.*, "Cloud computing: An overview," in *Proc. of the CLOUD'09*.

[2] M. Copeland *et al.*, "Microsoft azure and cloud computing," in *Microsoft Azure*. Springer, 2015, pp. 3–26.

[3] M. Sathiamoorthy *et al.*, "Xoring elephants: Novel erasure codes for big data," in *Proc. of the VLDB'13*.

[4] S. Muralidhar *et al.*, "f4: Facebook's warm blob storage system," in *Proc. of the OSDI'14*.

[5] K. Rashmi *et al.*, "A "hitchhiker's" guide to fast and efficient data reconstruction in erasure-coded data centers," in *Proc. of the Sigcomm'14*.

[6] Y. Zhang *et al.*, "Pcm: A parity-check matrix based approach to improve decoding performance of xor-based erasure codes," in *Proc. of the SRDS'15*.

[7] J. Gu *et al.*, "Optimizing the parity check matrix for efficient decoding of rs-based cloud storage systems," in *Proc. of the IPDPS'19*.

[8] M. Blaum *et al.*, "EVENODD: An efficient scheme for tolerating double disk failures in RAID architectures," *IEEE Transactions on Computers*, vol. 44, no. 2, pp. 192–202, 1995.

[9] P. Corbett *et al.*, "Row-Diagonal Parity for double disk failure correction," in *Proc. of the FAST'04*.

[10] C. Wu *et al.*, "HDP code: A Horizontal-Diagonal parity code to optimize I/O load balancing in RAID-6," in *Proc. of the DSN'11*.

[11] L. Xu *et al.*, "X-Code: MDS array codes with optimal encoding," *IEEE Transactions on Information Theory*, vol. 45, no. 1, pp. 272–276, 1999.

[12] C. Wu *et al.*, "H-Code: A hybrid MDS array code to optimize partial stripe writes in RAID-6," in *Proc. of the IPDPS'11*.

[13] Y. Zhang *et al.*, "Tip-code: A three independent parity code to tolerate triple disk failures with optimal update complextiy," in *Proc. of the DSN'15*.

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

[15] C. Huang *et al.*, "Erasure coding in windows azure storage." in *Proc. of the ATC'12*.

[16] M. Vajha *et al.*, "Clay codes: Moulding mds codes to yield an msr code," in *Proc. of the FAST'18*.

[17] T. Zhou and C. Tian, "Fast erasure coding for data storage: A comprehensive study of the acceleration techniques," in *Proc. of the FAST'19*.

[18] M. Xia *et al.*, "A tale of two erasure codes in hdfs," in *Proc. of the FAST'15*.

[19] H. Jin *et al.*, "Approximate code: A cost-effective erasure coding framework for tiered video storage in cloud systems," in *Proc. of the ICPP'19*.

[20] M. Ye and A. Barg, "Explicit constructions of optimal-access mds codes with nearly optimal sub-packetization," *IEEE Transactions on Information Theory*, vol. PP, no. 99, pp. 1–1, 2017.

[21] Y. Hu *et al.*, "Analysis and construction of functional regenerating codes with uncoded repair for distributed storage systems," in *Proc. of the INFOCOM'13*.

[22] L. Ye *et al.*, "Hybrid-rc: Flexible erasure codes with optimized recovery performance and low storage overhead," in *Proc. of the SRDS'17*.

[23] S. Kadekodi *et al.*, "Cluster storage systems gotta have heart: improving storage efficiency by exploiting disk-reliability heterogeneity," in *Proc. of the FAST'19*.

[24] Y. Tang *et al.*, "Mics: Mingling chained storage combining replication and erasure coding," in *Proc. of the SRDS'15*.

[25] H. Zhang *et al.*, "Efficient and available in-memory kv-store with hybrid erasure coding and replication," in *Proc. of the FAST'16*.

[26] L. Tian *et al.*, "Pro: A popularity-based multi-threaded reconstruction optimization for raid-structured storage systems." in *Proc. of the FAST'07*.

[27] S. Wu *et al.*, "Workout: I/o workload outsourcing for boosting raid reconstruction performance." in *Proc. of the FAST'09*.

[28] S. Wan *et al.*, "Victim disk first: an asymmetric cache to boost the performance of disk arrays under faulty conditions," in *Proc. of the ATC'13*.

[29] L. Li *et al.*, "Favorable block first: A comprehensive cache scheme to accelerate partial stripe recovery of triple disk failure tolerant arrays," in *Proc. of the ICPP'17*.

[30] J. Plank and L. Xu, "Optimizing Cauchy Reed-Solomon codes for Fault-Tolerant network storage applications," in *Proc. of the NCA'06*.

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

[32] M. Li *et al.*, "GRID codes: Strip-based erasure code with high fault tolerance for storage systems," *ACM Trans. on Storage*, vol. 4, no. 4, p. Article 15, Jan. 2009.

[33] Z. Shen and J. Shu, "HV code: An all-around mds code to improve efficiency and reliability of RAID-6 systems," in *Proc. of the DSN'14*.

[34] C. Wu *et al.*, "Code 5-6: An efficient mds array coding scheme to accelerate online raid level migration," in *Proc. of the ICPP'15*.

[35] X. Xie *et al.*, "Az-code: An efficient availability zone level erasure code to provide high fault tolerance in cloud storage systems," in *Proc. of the MSST'19*.

[36] K. Rashmi *et al.*, "Ec-cache: Load-balanced, low-latency cluster caching with online erasure coding," in *Proc. of the OSDI'16*.

[37] B. Fan *et al.*, "Diskreduce: Raid for data-intensive scalable computing," in *Proc. of the PDS'09*.

[38] R. Li, Y. Hu, and P. P. Lee, "Enabling efficient and reliable transition from replication to erasure coding for clustered file systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 9, pp. 2500–2513, 2017.

[39] P. K. Gummadi *et al.*, "Measurement, modeling, and analysis of a peer-to-peer file-sharing workload," in *Proc. of the SOSP'03*.

[40] S. Kavalanekar *et al.*, "Characterization of storage workload traces from production windows servers," in *Proc. of the IISWC'08*.

[41] L. Bairavasundaram *et al.*, "An analysis of latent sector errors in disk drives," in *Proc. of the SIGMETRICS'07*.

[42] A. Oprea and A. Juels, "A Clean-Slate look at disk scrubbing," in *Proc. of the FAST'10*.

[43] E. Pinheiro *et al.*, "Failure trends in a large disk drive population." in *Proc. of the FAST'07*.

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