Exploiting Pipelined Encoding Process to Boost Erasure-Coded Data Archival

Jianzhong Huang, Yanqun Wang, Xiao Qin, *Senior Member, IEEE*, Xianhai Liang, Shu Yin, and Changsheng Xie, *Member, IEEE*

Abstract—This paper addresses an issue of *erasure-coded data archival*, where $(k+r, k)$ erasure codes are employed to archive rarely accessed replicas. The traditional *synchronous encoding* process neither leverages the existence of replicas, nor handles encoding operations in a decentralized manner. To overcome these drawbacks, we exploit pipelined encoding processes to boost the data archival performance on storage clusters. First, we propose two data layouts called $[\mathbf{D}+\mathbf{P}]_{\mathbf{cd}}$ and $[\mathbf{3X}]_{\mathbf{cd}}$ by applying a chained-declustering mechanism to both *Mirrored RAID-5* and *triplication* redundancy groups. Second, in light of the $[\mathrm{D}+\mathrm{P}]_{\mathrm{cd}}$ and $[3\mathrm{X}]_{\mathrm{cd}}$ layouts, we design two archiving schemes named $\mathbf{DP}$ and $\mathbf{3X}$, which exhibit the following three salient features: (i) exploiting data locality—two or three local blocks are read by each involved node for encoding; (ii) decentralized computation load—encoding operations are distributed among *k* nodes; and (iii) parallel archival processing—two or three encoding pipelines are simultaneously deployed to generate parity blocks. We implement both the *DP* and *3X* schemes and three existing solutions (i.e., SynE, DE, and RapidRAID) in a real-world storage cluster. Experimental results show that our archival schemes outperform the other three solutions in terms of archiving time by a factor of at least 3.41 in a nine-node storage cluster. The experiments strongly indicate that the performance bottleneck of SynE lies in its block-receiving stage; it is disk I/O rather than network traffic that dominates archiving time for both the DE and RapidRAID schemes.

Index Terms—Erasure-coded storage cluster, data archival, pipelined encoding, power efficiency

1 Introduction

Data redundancy achieves high reliability by either replicating data blocks or storing additional information (e.g., parity blocks generated by erasure codes). The 3X-replica redundancy is employed by distributed storage systems (e.g., GFS [1] and HDFS [2], and Amazon S3 [3]) to keep data durable. Compared to data replication, erasure codes provide equivalent fault-tolerance with significantly small storage overhead [4]. Most of the data are accessed within a short duration of the data's lifetime. For example, over 90 percent of accesses in a Yahoo! M45 Hadoop cluster occur within the first day after data creation [5]; therefore, it is economically friendly to archive data replicas using erasure codes. Nowadays, some real-world storage systems (e.g., WAS [6], GFS II [7]) adopt a hybrid redundancy strategy, where a replication strategy is applied to newly created data, while erasure codes are used to archive the same data once its access frequency decreases.

With *Synchronous Encoding* (or *SynE* for short) [5], parity blocks are generated using classical erasure codes. If a $(k+r, k)$ erasure code is used, an encoding node retrieves k different blocks from existing data replicas; then the node computes and delivers the resulting r parity blocks to r other different nodes. The traffic incurred by parity generation and parity migration is k blocks and r blocks, respectively. The parity generation traffic could be reduced to $k-1$ blocks if the encoding operation is conducted by a node storing data replicas. Such a centralized encoding process makes the encoding node become a performance bottleneck of data archival.

Apart from *SynE*, two families of new erasure codes (i.e., DE [8] and RapidRAID [9]) were proposed to address the issue of erasure-coded data archival. Both DE and RapidRAID accomplish the parity generation through a decentralized encoding process. The downside of DE and RapidRAID is that each involved encoding node has to service both read and write requests, thereby decreasing write bandwidth due to bandwidth competition between reads and writes. The degraded write bandwidth inevitably deteriorates archival performance.

The overarching goal of this study is to speed up the archival process by leveraging the locality of replicas during the encoding procedure. To achieve this goal, we focus on a data layout strategy that places newly created data; we show how such a layout allows data archival to capture the following two features: (1) to accomplish data archival in a pipelined manner—distributing encoding processes among multiple nodes to form an encoding pipeline, which alleviates the performance bottleneck in centralized encoding; and (2) to increase archival parallelism—allowing all the nodes to form two or more encoding pipelines, which speed up the archival performance.

We propose two data layouts—$[\mathrm{D}+\mathrm{P}]_{\mathrm{cd}}$ and $[3\mathrm{X}]_{\mathrm{cd}}$—by applying the chained-declustering (CD) mechanism to the

- J. Huang, Y. Wang, X. Liang, and C. Xie are with Wuhan National Lab. for Optoelectronics, Huazhong University of Science and Technology, Wuhan 430074, China. E-mail: {hjzh, cs_xie}@hust.edu.cn, {wychktk2013, hustlxh}@gmail.com.
- X. Qin is with the Department of Computer Science and Software Engineering, Shelby Center for Engineering Technology, Samuel Ginn College of Engineering, Auburn University, AL 36849-5347. E-mail: xqin@auburn.edu.
- S. Yin is with the Hunan University, China. E-mail: 16shuyin@gmail.com.

Manuscript received 16 June 2014; revised 17 Sept. 2014; accepted 27 Oct. 2014. Date of publication 30 Oct. 2014; date of current version 7 Oct. 2015. Recommended for acceptance by A. R. Butt.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2014.2366113

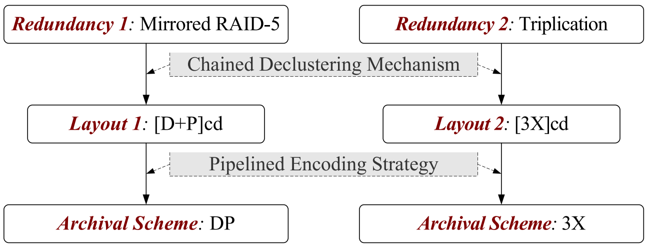

Redundancy 1: Mirrored RAID-5
Redundancy 2: Triplication
Chained Declustering Mechanism
Layout 1: [D+P]cd
Layout 2: [3X]cd
Pipelined Encoding Strategy
Archival Scheme: DP
Archival Scheme: 3X

Fig. 1. The chained-declustering mechanism is applied to two redundancy groups—Mirrored RAID-5 and Triplication—to yield two layouts, i.e., $[\mathrm{D+P}]_{\mathrm{cd}}$ (see Section 2.1) and $[3\mathrm{X}]_{\mathrm{cd}}$ (see Section 2.2). The pipelined encoding strategy is incorporated into the two layouts to design two archival schemes, i.e., DP (see Section 3.1) and 3X (see Section 3.2).

data organized in Mirrored RAID-5 and triplication forms, respectively. Two erasure-coded archival schemes called DP and 3X are designed by applying the $[D+P]_{cd}$ and $[3X]_{cd}$ data layouts, respectively. Fig. 1 outlines the relationship between the two layouts and the two archival schemes implemented in light of pipelined encoding strategy.

SynE deploys a centralized encoding process, which is likely to deteriorate encoding performance. Different from *SynE*, the DP and 3X schemes distribute encoding operations among multiple nodes. Unlike DE and RapidRAID schemes that suffer from bandwidth competition that slows down archival process, DP and 3X judiciously maximize disk bandwidth by making each node respond to either reads or writes. The archival performance of DP and 3X is dominated by network I/Os.

The contributions of this study are summarized as follows:

- We introduce the chained-declustering mechanism to both Mirrored RAID-5 and triplication redundancy groups to reach two new data layouts $[D+P]_{cd}$ and $[3X]_{cd}$, respectively. Both data layouts not only have potential in power efficiency, but also allow a subset of involved nodes to constitute an encoding pipeline.
- We develop two archival schemes (i.e., *DP* and *3X*), which respectively apply the $[D+P]_{cd}$ and $[3X]_{cd}$ data layouts to improve performance. Both archival schemes exploit data locality of local replicas while decentralizing the encoding process. Additionally, all the nodes in *DP* and *3X* collaborate to form multiple (e.g., two or three) encoding pipelines to facilitate archival parallelism.
- We implement the *DP* and *3X* schemes as well as three existing approaches (i.e., *SynE*, *DE*, and *RapidRAID*) in a real-world storage cluster. Our experiments show that the proposed archival schemes outperform the other three solutions by a factor of up to 3.41 in a nine-node storage cluster. Interestingly, our findings reveal that the performance bottleneck of *SynE* lies in its block-receiving stage; it is disk I/Os that dominate the archival performance in the *DE* and *RapidRAID* schemes rather than network I/Os.

The rest of the paper is organized as follows. In Section 2 we present two chained-declustering-based layouts, namely, $[D+P]_{cd}$ and $[3X]_{cd}$. Section 3 details our pipelined archival schemes—*DP* and *3X*. Section 4 provides the comparative analysis of the five archival schemes. We describe the experiments in Section 5. Sections 6 and 7 discuss related work and applicability issues, respectively. Finally, we conclude this paper in Section 8.

TABLE 1
Layout of $[\mathrm{D+P}]_{\mathrm{cd}}$

Node:	N_1	N_2	N_3	$\cdots$	N_k	N_{k+1}
Primary:	d_1	d_2	d_3	$\cdots$	d_k	p_1
Replica:	p_1	d_1	d_2	$\cdots$	d_{k-1}	d_k

2 Chained-Declustering-Based Layouts

Prior studies (e.g., chained declustering [10], group rotational declustering [11], and shifted declustering [12]) suggest that a declustering strategy is able to improve I/O parallelisms for replication-based storage. CD is a simple yet efficient policy (i.e., it simply shifts each row of data units in a circular fashion); we introduce CD to *Mirrored RAID-5* and *triplication* to leverage data locality for efficient data archival.

2.1 Applying the CD Mechanism to Mirrored RAID-5

Compared to triplication, '*Mirrored RAID-5*' [13], '*Mirror+P*' [14], and '*RAID 5+Mirror*' [15] provide better tradeoffs between system availability and storage efficiency. We advocate *Mirrored RAID-5* in the context of storage clusters; therefore, let us describe a chained-declustering-based '*Mirrored RAID-5*' layout called $[\mathrm{D+P}]_{\mathrm{cd}}$.

2.1.1 Layout of [D+P]$_{cd}$

As depicted in Table 1, the $[\mathrm{D+P}]_{\mathrm{cd}}$ layout makes $k+1$ blocks $\{\mathrm{d}_1, \mathrm{d}_2, \ldots, \mathrm{d}_k, \mathrm{p}_1\}$ be dispersed among $k+1$ nodes $\{N_1, N_2, \ldots, N_{k+1}\}$ in a chained-declustering manner, with $\mathrm{p}_1 = \mathrm{d}_1 \oplus \mathrm{d}_2 \oplus \cdots \oplus \mathrm{d}_k$. All the $k+1$ nodes are conceptually organized in a chain, in which a primary block and its replica are placed in adjacent nodes.

The placement of replicas is critical to data reliability; the rack-aware replica placement policy in HDFS is a good example [16]. With the rack-aware replica placement policy in place, one replica is stored on one node in a local rack, another replica is placed on a node in a remote rack, and the last one is on a separate node in the same remote rack. When it comes to layout $[\mathrm{D+P}]_{\mathrm{cd}}$, at most two nodes out of a node chain are placed in the same rack; this policy does not compromise data accessibility even in the presence of failures of two nodes or an entire rack.

2.1.2 Supporting Double-Fault Tolerance

We consider a generic chained-declustering layout (see Table 2), in which the ith primary block $\mathrm{b}_\mathrm{i}^\mathrm{p}$ is placed on node N_i, and its replica $\mathrm{b}_\mathrm{i}^\mathrm{r}$ is placed on the *next* node $N_{(\mathrm{i+1 mod n})}$.

Definition 1. *Let us consider a block collection $b = \{b_1, b_2, \ldots, b_n\}$, where $b_i = \{b_i^p, b_i^r\}$, $1 \le i \le n$. If primary block b_i^p survives, then the step function $s(b_i^p)$ returns 1; otherwise, $s(b_i^p)$ returns 0. Similarly, the step function $s(b_i^r)$ returns 1 if*

TABLE 2
An n-Node Chained Declustering Ring

Node:	N_1	N_2	N_3	$\cdots$	N_{n-1}	N_n
Primary:	b_1^p	b_2^p	b_3^p	$\cdots$	b_{n-1}^p	b_n^p
Replica:	b_n^r	b_1^r	b_2^r	$\cdots$	b_{n-2}^r	b_{n-1}^r

TABLE 3
A 7-Node $[D{+}P]_{cd}$ with Four Failures

Node:	N_1	N_2	N_3	N_4	N_5	N_6	N_7
Primary	—	d_2	—	d_4	—	d_6	—
Replica	—	d_1	—	d_3	—	d_5	—

replica block b_i^r survives; otherwise, $s(b_i^r)$ returns 0. The survive function $s(b_i)$ can be derived from $s(b_i^p)$ and $s(b_i^r)$ as Eq. (1):

$$s(b_i) = s(b_i^p) \parallel s(b_i^r) \quad i \in \{1, 2, \ldots, n\}. \tag{1}$$

Let $\mathrm{num}(b)$ be the number of *essential* surviving blocks in collection b. $\mathrm{num}(b)$ is expressed as Eq. (2):

$$\mathrm{num}(b) = s(b_1) + s(b_2) + \cdots + s(b_n). \tag{2}$$

Lemma 1. *There exist at least $n-1$ essential surviving blocks (i.e., num(b) $\geq n-1$) when two nodes fail.*

Proof. If two adjacent nodes N_i and N_{i+1} fail, then $n-1$ different blocks $\{b_1^p, b_2^p, \ldots, b_{i-1}^p, b_{i+1}^r, b_{i+2}^p, \ldots, b_n^p\}$ survive. Thus, we have 'num(b) = n−1'. On the other hand, if two failed nodes N_i and N_{i+2} are non-adjacent, then n different blocks $\{b_1^p, b_2^p, \ldots, b_{i-1}^p, b_i^r, b_{i+1}^p, b_{i+2}^r, b_{i+3}^p, \ldots, b_n^p\}$ survive. Therefore, equation 'num(b) = n' also holds. This concludes the proof of Lemma 1. □

Lemma 1 suggests that there are at least k essential surviving blocks for $[D+P]_{cd}$ in the double-node-failure case. In other words, layout $[D+P]_{cd}$ can tolerate double node failures.

2.1.3 Achieving Power Efficiency

As to fault-tolerant storage systems, redundant devices are logically envisioned as erased ones that can be deactivated to conserve energy [17], [18]. Layout $[D+P]_{cd}$ offers double-node-fault tolerance, thereby allowing two nodes to be placed into low-power mode to reduce energy consumption and the rest of the nodes to be active to respond to user I/O requests.

We introduce two access policies to avoid unnecessary power-state transitions in the offline nodes.

- *Read policy*. Reads in the energy-saving mode are handled as below: (1) If a requested data block is residing on an active node, then the read request is directly serviced, and no extra computation overhead occurs; (2) otherwise, k essential blocks are retrieved from the active nodes, followed by calculation of the requested data block from the k blocks. In the latter case, the requested data block and its replica are residing on two adjacent offline nodes, and parity block p_1 exists in the k retrieved essential blocks.
- *Write policy*. In archival storage systems, it is critical to address the issue of writing newly created data blocks rather than updating existing ones. Data writes in the energy-efficient mode are processed as follows. A replica of new data blocks are stored to the remaining active nodes according to the $[D+P]_{cd}$ layout. When the inactive nodes are activated, a replica of each new data block is written to the activated nodes; in addition, two copies of the corresponding parity block are placed to the activated nodes.

2.1.4 Analysis of Fault Tolerance

As mentioned in Section 2.1.2, layout $[D+P]_{cd}$ can tolerate double node failures. Furthermore, $[D+P]_{cd}$ can tolerate $\lceil (k+1)/2 \rceil$ failures as long as surviving nodes are not adjacent (see, for example, Table 3). $\lceil (k+1)/2 \rceil$-fault-tolerance—the best case for the $[D+P]_{cd}$ layout—can be applied to optimize power efficiency. This is because $\lceil (k+1)/2 \rceil$ nodes can be placed into the low-power mode.

It is worth noting that the $[D+P]_{cd}$ layout cannot always tolerate $\lceil (k+1)/2 \rceil$ node failures, because the positions of node failures are unpredictable. It is of necessity to consider how to handle node failures when $\lceil (k+1)/2 \rceil$ nodes stay in the inactive/low-power mode. A basic principle is to power up a certain number of inactive nodes. Let us consider a straightforward approach given as follows. When an active node (e.g., N_2 in Table 3) fails, one can simply activate two inactive nodes (e.g., N_1 and N_3 in Table 3) adjacent to the failed one. This is because that a primary block and its replica are placed in adjacent nodes. In this case, the two activated nodes can be used to recover the failed node while responding to user I/O requests.

The aforementioned approach still takes effect in the double-node-failure case. Of course, it does not make sense to achieve power efficiency at the cost of data loss. Therefore, a node-reconstruction process should be immediately triggered once two nodes concurrently fail.

2.2 Applying the CD Mechanism to Triplication

In this section, we describe a data layout called $[3X]_{cd}$, where the chained-declustering mechanism is applied to triplication—the de facto redundancy scheme widely employed by production storage systems.

Table 4 illustrates the layout of $[3X]_{cd}$ for k data blocks $\{d_1, d_2, \ldots, d_k\}$, where all k nodes $\{N_1, N_2, \ldots, N_k\}$ are conceptually organized in a chain; a primary block and its two replicas are placed in any three adjacent nodes in the chain.

As to any data block d_i (for, $i \in \{1, 2, \ldots, k\}$), at least one out of three copies of block d_i survives in the presence of

TABLE 4
Layout of $[3X]_{cd}$

Node:	N_1	N_2	N_3	$\cdots$	N_{k-1}	N_k
Primary:	d_1	d_2	d_3	$\cdots$	d_{k-1}	d_k
First Replica:	d_k	d_1	d_2	$\cdots$	d_{k-2}	d_{k-1}
Second Replica:	d_{k-1}	d_k	d_1	$\cdots$	d_{k-3}	d_{k-2}

TABLE 5
A Six-Node $[3\mathrm{X}]_{\mathrm{cd}}$ with Four Failures

Node:	N_1	N_2	N_3	N_4	N_5	N_6
Primary	d_1	—	—	d_4	—	—
First Replica	d_6	—	—	d_3	—	—
Second Replica	d_5	—	—	d_2	—	—

two-node failures. In other words, there are exactly k essential surviving blocks when two nodes fail; and thus, layout $[3\mathrm{X}]_{\mathrm{cd}}$ is able to tolerate double-node failures. More interestingly, $[3\mathrm{X}]_{\mathrm{cd}}$ can tolerate $k - \lceil k/3 \rceil$ node failures when there are two faulty nodes sitting between two surviving nodes. For example, the six-node $[3\mathrm{X}]_{\mathrm{cd}}$ layout plotted in Table 5 still offers $k = 6$ essential surviving blocks in the four-node-failure case (i.e., four faulty nodes are N_2, N_3, N_5, and N_6). Like $[\mathrm{D}+\mathrm{P}]_{\mathrm{cd}}$, the $[3\mathrm{X}]_{\mathrm{cd}}$ layout should address the issue of tackling node-failure problems when $k - \lceil k/3 \rceil$ nodes still stay in the low-power mode. Similarly, the node-activation approach in $[\mathrm{D}+\mathrm{P}]_{\mathrm{cd}}$ can be applied to the $[3\mathrm{X}]_{\mathrm{cd}}$ layout as follows. If an active node fails, one may simply activate two inactive nodes adjacent to the failed node.

The read/write policies of $[3\mathrm{X}]_{\mathrm{cd}}$ in the energy-saving mode are different from those of $[\mathrm{D}+\mathrm{P}]_{\mathrm{cd}}$ because the $[3\mathrm{X}]_{\mathrm{cd}}$ layout does not include a parity block (e.g., p_1 in $[\mathrm{D}+\mathrm{P}]_{\mathrm{cd}}$).

- *Read policy*. Reads in the energy-efficient mode are handled as follows. If a requested data block is residing on an active node, then the read request is directly serviced; otherwise, an offline node holding the requested data block will be activated to respond to the read request.
- *Write policy*. Data writes in the energy-efficient mode are processed as follows. A replica of new data blocks are stored to the remaining active nodes according to the $[3\mathrm{X}]_{\mathrm{cd}}$ layout. When inactive nodes are activated, two replicas of each new data block will be placed to the activated nodes accordingly.

3 Pipelined Data Archival

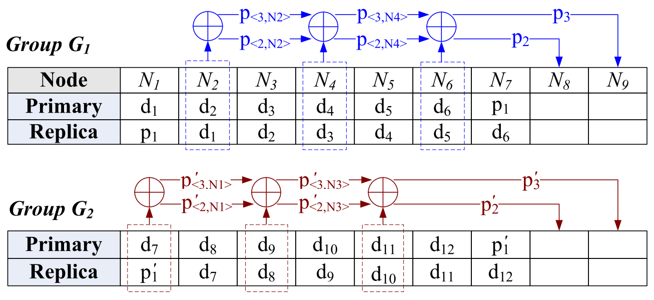
3.1 Archival for the [D+P]$_{\mathrm{cd}}$ Layout

It is a conventional wisdom that if data replicas are no longer modified, the replicas can be erasure coded to save storage space. Such an erasure coding process using existing data is referred to as '*erasure-coded data archival*'. There are four steps involved in data archival within storage clusters: (1) retrieving required data blocks; (2) performing an encoding over the retrieved blocks; (3) migrating resulting parity blocks to separate nodes; and (4) deleting the replica blocks.

Since numerous production storage systems adopt Reed Solomon (RS) codes [19], [20] to store infrequently accessed data [21], [7], [22], our archival scheme—*DP*—employs RS codes to archive replicas stored in a $[\mathrm{D}+\mathrm{P}]_{\mathrm{cd}}$ manner.

According to the RS encoding procedure [23], parity block p_j can be derived as Eq. (3), where $\alpha_{\mathrm{j,i}}$ is a coefficient in the generator matrix of RS codes:

$$\mathrm{p_j} = \alpha_{\mathrm{j,1}}\mathrm{d_1} + \alpha_{\mathrm{j,2}}\mathrm{d_2} + \cdots + \alpha_{\mathrm{j,k}}\mathrm{d_k}. \tag{3}$$



Node	N_1	N_2	N_3	N_4	N_5	N_6	N_7	N_8	N_9
Primary	d_1	d_2	d_3	d_4	d_5	d_6	p_1		
Replica	p_1	d_1	d_2	d_3	d_4	d_5	d_6		

Primary	d_7	d_8	d_9	d_{10}	d_{11}	d_{12}	p'_1		
Replica	p'_1	d_7	d_8	d_9	d_{10}	d_{11}	d_{12}		

Fig. 2. The DP archival scheme—data stored in the $[D+P]_{cd}$ layout is migrated to erasure-coded storage, with coding parameters $k = 6$ and $r = 3$. Symbol '$\oplus$' denotes the linear combination.

Given $k = 6$, Eq. (3) is decomposed as

$$\mathrm{p}_{<\mathrm{j},\mathrm{N}_2>} = \alpha_{\mathrm{j,1}}\mathrm{d_1} + \alpha_{\mathrm{j,2}}\mathrm{d_2}, \tag{3.1}$$

$$\mathrm{p}_{<\mathrm{j},\mathrm{N}_4>} = \mathrm{p}_{<\mathrm{j},\mathrm{N}_2>} + \alpha_{\mathrm{j,3}}\mathrm{d_3} + \alpha_{\mathrm{j,4}}\mathrm{d_4}, \tag{3.2}$$

$$\mathrm{p_j} = \mathrm{p}_{<\mathrm{j},\mathrm{N}_4>} + \alpha_{\mathrm{j,5}}\mathrm{d_5} + \alpha_{\mathrm{j,6}}\mathrm{d_6}. \tag{3.3}$$

Parity block $\mathrm{p_j}$ $(2 \le j \le r)$ can also be calculated from k blocks, one of which is parity block $\mathrm{p_1}$. The k blocks are $\{\mathrm{d}_1, \mathrm{d}_2, \ldots, \mathrm{d}_{\mathrm{k}-1}, \mathrm{p}_1\}$ or $\{\mathrm{d}_2, \mathrm{d}_3, \ldots, \mathrm{d}_{\mathrm{k}}, \mathrm{p}_1\}$. When k equals to 6, block $\mathrm{p_j}$ is expressed as Eq. (4):

$$\begin{aligned}\mathrm{p_j} = {} & (\alpha_{\mathrm{j,1}} - \alpha_{\mathrm{j,6}})\mathrm{d_1} + \alpha_{\mathrm{j,6}}\mathrm{p_1} + (\alpha_{\mathrm{j,2}} - \alpha_{\mathrm{j,6}})\mathrm{d_2} + (\alpha_{\mathrm{j,3}} - \alpha_{\mathrm{j,6}})\mathrm{d_3} \\ & + (\alpha_{\mathrm{j,4}} - \alpha_{\mathrm{j,6}})\mathrm{d_4} + (\alpha_{\mathrm{j,5}} - \alpha_{\mathrm{j,6}})\mathrm{d_5}.\end{aligned} \tag{4}$$

Similarly, Eq. (4) can be decomposed into the following three expressions:

$$\mathrm{p}_{<\mathrm{j},\mathrm{N}_1>} = (\alpha_{\mathrm{j,1}} - \alpha_{\mathrm{j,6}})\mathrm{d_1} + \alpha_{\mathrm{j,6}}\mathrm{p_1}, \tag{4.1}$$

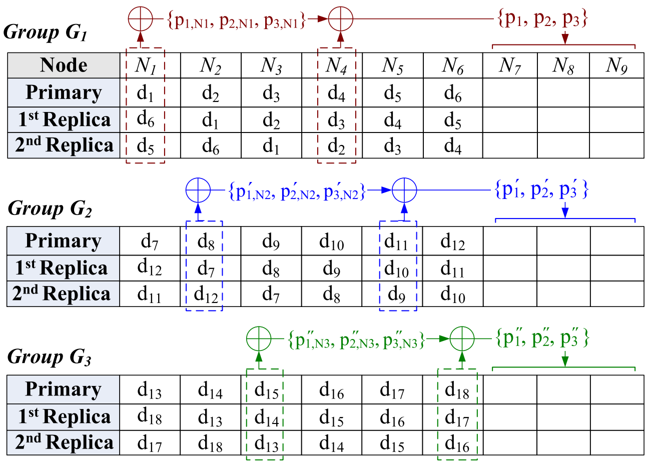
$$\mathrm{p}_{<\mathrm{j},\mathrm{N}_3>} = \mathrm{p}_{<\mathrm{j},\mathrm{N}_1>} + (\alpha_{\mathrm{j,2}} - \alpha_{\mathrm{j,6}})\mathrm{d_2} + (\alpha_{\mathrm{j,3}} - \alpha_{\mathrm{j,6}})\mathrm{d_3}, \tag{4.2}$$

$$\mathrm{p_j} = \mathrm{p}_{<\mathrm{j},\mathrm{N}_3>} + (\alpha_{\mathrm{j,4}} - \alpha_{\mathrm{j,6}})\mathrm{d_4} + (\alpha_{\mathrm{j,5}} - \alpha_{\mathrm{j,6}})\mathrm{d_5}. \tag{4.3}$$

Since storage nodes offer sufficient Reed Solomon coding capability [24], [25], [26], [27], nodes N_2, N_4, and N_6 are able to compute the linear combinations using Eqs. (3.1)-(3.3). For example (see group G_1 in Fig. 2), node N_2 generates $r - 1 = 2$ intermediate parity blocks $\mathrm{p}_{<2,\mathrm{N}_2>}$ and $\mathrm{p}_{<3,\mathrm{N}_2>}$, using two local blocks $\mathrm{d_1}$ and $\mathrm{d_2}$ according to Eq. (3.1), and delivers the intermediate parity blocks to node N_4. Similarly, node N_6 computes two parity blocks $\mathrm{p_2}$ and $\mathrm{p_3}$ for nodes N_8 and N_9 using Eq. (3.3).

As shown in Fig. 2, the three non-adjacent nodes N_2, N_4, and N_6 in group G_1 form an encoding pipeline '$N_2 \rightarrow N_4 \rightarrow N_6$'. Similarly, a second encoding pipeline '$N_1 \rightarrow N_3 \rightarrow N_5$' is constituted from three non-adjacent nodes N_1, N_3, and N_5 in group G_2. Simultaneously applying the two encoding pipelines to two separate replica sets can accomplish parallel archival processes.

When only one encoding pipeline is deployed to handle data archival, one may deactivate $\lceil (k+1)/2 \rceil$ nodes into the power-saving mode to conserve energy. We refer to the DP scheme in such a power-efficient mode as $\mathrm{DP}_{\mathrm{power}}$.



Group G1
{p1,N1, p2,N1, p3,N1}
{p1, p2, p3}
Node | N1 | N2 | N3 | N4 | N5 | N6 | N7 | N8 | N9
Primary | d1 | d2 | d3 | d4 | d5 | d6
1st Replica | d6 | d1 | d2 | d3 | d4 | d5
2nd Replica | d5 | d6 | d1 | d2 | d3 | d4
Group G2
{p′1,N2, p′2,N2, p′3,N2}
{p′1, p′2, p′3}
Primary | d7 | d8 | d9 | d10 | d11 | d12
1st Replica | d12 | d7 | d8 | d9 | d10 | d11
2nd Replica | d11 | d12 | d7 | d8 | d9 | d10
Group G3
{p″1,N3, p″2,N3, p″3,N3}
{p″1, p″2, p″3}
Primary | d13 | d14 | d15 | d16 | d17 | d18
1st Replica | d18 | d13 | d14 | d15 | d16 | d17
2nd Replica | d17 | d18 | d13 | d14 | d15 | d16

Fig. 3. The 3X archival scheme—data stored in the $3X_{cd}$ layout is migrated to erasure-coded storage. $k = 6$ and $r = 3$. All the data nodes are grouped into three groups, which simultaneously carry out the encoding processes for separate replica sets.

In a word, layout $[D + P]_{cd}$ offers opportunities for storage clusters to optimize the archival process from three aspects: (1) Exploiting data locality—two local blocks are read by each involved node for encoding; (2) decentralized computation load—unlike a single encoding node in the SynE scheme, $\lceil k/2 \rceil$ nodes constitute an entire encoding pipeline; (3) parallel archival—two encoding pipelines are simultaneously executed to yield parity blocks.

3.2 Archival for the $[3X]_{cd}$ Layout

Since 3X-replica redundancy is widely employed in production storage systems, it is practical and vital to study how to efficiently migrate from 3x replication to erasure coding. Apart from the DP scheme, an archival scheme called *3X* is proposed to archive replicas organized in the $[3X]_{cd}$ layout using RS codes.

According to data placement governed by layout $[3X]_{cd}$, all nodes can be divided into $\lfloor k/\lceil k/3 \rceil \rfloor$ groups, each of which handles encoding operations in a pipelined manner. Fig. 3 shows that the nodes holding six raw data blocks $\{d_1, d_2, \ldots, d_6\}$ are divided into three groups (i.e., G_1, G_2, and G_3). All the groups are able to simultaneously carry out encoding processes for separate replica sets, thereby enabling high-performance archival through increased archival parallelism.

Similar to DP_{power}, the $3X_{power}$ archival scheme deploys an encoding pipeline to generate parity blocks from raw data blocks organized in the $[3X]_{cd}$ layout. In $3X_{power}$, $\lceil k/3 \rceil$ nodes are responsible for the redundancy generation; $k - \lceil k/3 \rceil$ nodes are deactivated to save power.

3.3 I/O Optimization for DP and 3X

3.3.1 *Write Aggregation*

Block sizes of most existing cluster file systems are large. For example, the default block size is 64 MB in GFS [1], QFS [28], and HDFS [2], and it can be configured to 128, 256, 512 MB, etc. A storage request unit (SRU) is a basis access unit from the I/O path's standpoint; hence, a block is accessed via a sequence of SRUs. I/O accesses are likely to be non-

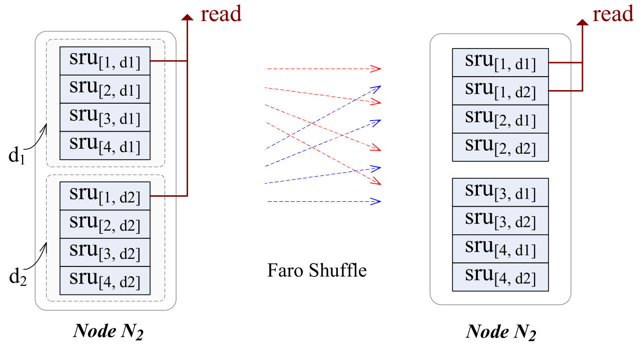


Fig. 4. The Faro shuffle algorithm is employed to handle Logical-to-Physical address mapping for SRUs in blocks within a node.

sequential when requested SRUs are located on different blocks, thereby degrading the available write bandwidth.

Write bandwidth degradation might occur in the DP scheme if the following two conditions are met: (1) Two parity blocks generated by two encoding pipelines are written to an identical node; (2) these two parity blocks are placed to different disk regions. Motivated by the fact that large SRU requests help in achieving high write bandwidth [29], [30], [31], we introduce the *write aggregation* technique to both DP and 3X schemes. The write aggregation technique consolidates multiple SRUs into a single write request. In particular, the node storing parity blocks pre-allocates a memory region to buffer SRUs delivered from different encoding pipelines, then writes the buffered SRUs in the form of a large sequential write.

3.3.2 *Shuffle Mapping*

In the DP scheme, one encoding node must read two SRUs from two data blocks. Such a reading pattern results in non-sequential I/Os if the two blocks are residing in the same disk. To solve this problem, we map a range of logical SRUs to physical locations using the faro shuffle algorithm [32]. The faro shuffle is equivalent to a cyclic logical left shift in terms of binary representation. An example is shown in Fig. 4b, where the first SRU $sru_{[1,d_1]}$ in data block d_1 and the first SRU $sru_{[1,d_2]}$ in data block d_2 are contiguous. The space reservation function of the underlying file system can be used to contiguously write SRUs [28]; thus, two SRUs can be placed contiguously when storing newly created data blocks. By virtue of contiguous writes, each encoding node accomplishes high read sequentiality during data archival.

The shuffle mapping algorithm can be extended to shuffle SRUs in three blocks—three SRUs stored in three different blocks are contiguously placed in a disk. Therefore, the extended shuffle mapping algorithm can be deployed to improve the read bandwidth for the 3X scheme.

4 Comparative Analysis

4.1 Network Traffic

Fig. 2 shows that six blocks (i.e., four intermediate parity blocks and two final parity blocks) are delivered to accomplish the DP archival process. In a general case, network traffic induced by the DP scheme is $\lceil k/2 \rceil \times (r - 1)$ blocks;

TABLE 6
Comparisons of the (k+r,k) Erasure-Coded Data Archival Schemes

Scheme	Redundancy	Number of encoding nodes	Normalized network traffic
Synchronous Encoding [5]	3-way replication $\Rightarrow$ classical erasure codes	k	$k + r - 1$
DE [8]	$2 \sim 3$-way replication $\Rightarrow$ Decentralized erasure codes	r	$*$
RapidRAID [9]	2-way replication $\Rightarrow$ Pipelined erasure codes	$k + r$	$k + r - 1$
DP	Mirrored RAID-5 $\Rightarrow$ Reed Solomon codes	$2\times\lceil k/2\rceil$ for DP; $\lceil k/2\rceil$ for DP_{power}	$\lceil k/2\rceil\times(r-1)$
3X	3-way replication $\Rightarrow$ Reed Solomon codes	$(k/\lceil k/3\rceil)\times\lceil k/3\rceil$ for 3X; $\lceil k/3\rceil$ for $3X_{power}$	$\lceil k/3\rceil\times(r)$

$*$: *The traffic caused by data archival is a function of the number* l *of blocks stored in* r *coding nodes.*

on the other hand, the 3X scheme leads to network traffic of $\lceil k/3\rceil \times (r)$ blocks (see the comparison in Table 6).

The archival traffic in *SynE* is $k + r - 1$ blocks if a node keeping a data replica acts as the encoding node (see node N_6 in Fig. 5a). In the DE archival process, the network traffic is a function of the number l of blocks stored in r coding nodes, and the value of l ranges from k to $2k$. Fig. 5b depicts the network flow of DE, in which the value of l equals to k (viz., DE($l = k$)). It is revealed from Fig. 5c that the traffic incurred by the RapidRAID archival process is $k + r - 1$ blocks.

When coding parameters k and r are set to 6 and 3 (i.e., $k = 3$ and $r = 3$), the normalized traffic of SynE, DE, RapidRAID, DP, and 3X is eight, four, eight, six, and six blocks, respectively. Surprisingly, network traffic caused by the archival process has no noticeable impact on the archival performance (see Section 5.3.1).

4.2 Disk I/Os

Fig. 6 shows the data flows of five archival schemes (i.e., SynE, DE, RapidRAID, DP, and 3X). The data flow in each scheme is exemplified by both network transmission and disk I/Os of an encoding node and a parity node.

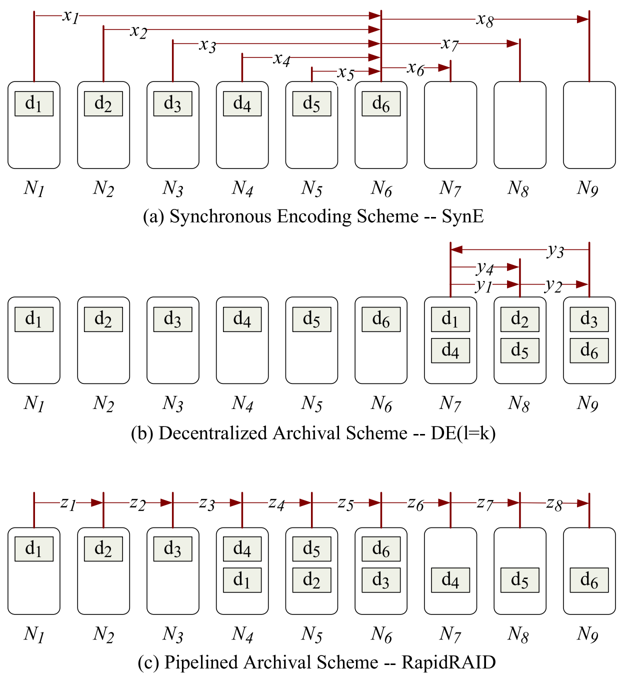

x1
x2
x3
x4
x5
x6
x7
x8
d1
d2
d3
d4
d5
d6
N1
N2
N3
N4
N5
N6
N7
N8
N9
(a) Synchronous Encoding Scheme -- SynE
y1
y2
y3
y4
(b) Decentralized Archival Scheme -- DE(l=k)
z1
z2
z3
z4
z5
z6
z7
z8
(c) Pipelined Archival Scheme -- RapidRAID

Fig. 5. Network flows in the three erasure-coded archival schemes, with coding parameters $k = 6$ and $r = 3$.

Recall that two types of disk I/O operations involved in the archival process are: (1) reading required data blocks, and (2) writing resulting encoded blocks. Figs. 6a, 6 d, and 6e illustrate that each node in SynE, DP, and 3X responds to either read requests or write requests.

When it comes to DE, l data blocks are spread among r coding nodes $\{N_{k+1}, N_{k+2}, \ldots, N_{k+r}\}$, meaning that one of the coding nodes has to read at least $\lceil l/r\rceil$ blocks. Apart from reads, each coding node has to write one encoded block. Thus, DE's write bandwidth is approximate to $1/(1+\lceil l/r\rceil)$ of the total available disk bandwidth. The available disk bandwidth might be reduced due to non-sequential accesses, which occur when existing data blocks and the encoded parity blocks are residing on different regions of a disk. We address this non-sequential-access issue in our experiments by introducing multiple disks to improve I/O sequentiality (see Section 5.3.4).

In RapidRAID, if parameter k is larger than parameter r, then each of $k - r$ nodes $\{N_{r+1}, N_{r+2}, \ldots, N_k\}$ will respond to two read requests and one write request (see Node N_4 in Fig. 6c). We evaluate the disk throughput by running IOMeter [33] on HDD disks deployed in the tested storage cluster (see details of the experimental environment in Section 5.1); we observe that the maximum throughput of a disk is anywhere between 120 to 135 MBps. Thus, we estimate that the available write bandwidth in RapidRAID is less than 135/3 = 45 MBps. The available sending/receiving bandwidth of each node is approximately 800 Mbps[1]. Analytically, disk I/Os rather than network I/Os become a performance bottleneck of the archiving process in the RapidRAID scheme.

5 Experimental Evaluation

We implement our DP and 3X schemes along with the three alternatives (i.e., SynE, DE, and RapidRAID) in a real-world storage cluster. We conduct extensive experiments to quantitatively compare the archival performance of the five solutions.

1. The sending/receiving bandwidth of bidirectional communication is lower than that of unidirectional communication (e.g., 800 versus 900 Mbps), since evidence shows that the performance ratio (i.e., a ratio between measured and theoretical bandwidth) of bidirectional communication is lower than that of unidirectional communication [34].

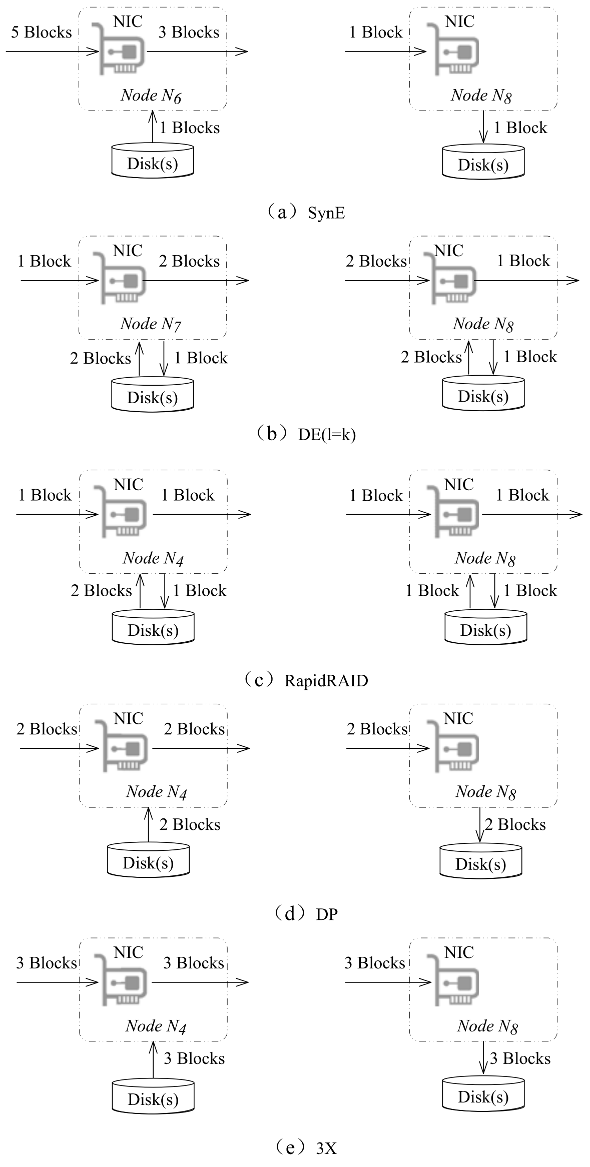


Fig. 6. Comparisons of the data flows in the five erasure-coded archival schemes (i.e., SynE, DE($l = k$), RapidRAID, DP, and 3X), with coding parameters $k = 6$ and $r = 3$.

5.1 Experimental Setup

Our testbed is a storage cluster that consists of 12 storage nodes connected through a Cisco GibE switch. Each storage node contains an Intel(R) E5-2650 @ 2.0 GHz CPU, 16 GB DDR3 memory, and Intel C600 Chipset Mainboard with 1 GbE integrated NIC. Four disks are attached to each node; all the disks are Western Digital's Enterprise WD1003FBYX SATA2.0 disks. The operating system running in the storage nodes is Ubuntu 10.04 X86 64 (Kernel 2.6.32).

5.2 Evaluation Methodology

Prior evidence shows that a configuration of '$r = 3$' achieves a sufficiently large mean-time-to-data-loss or MTTDL for archival storage [24]. Furthermore, r is set to 3 (i.e., $r = 3$) in GFS II [7], HydraStor [35], and QFS [28]; r is set to 4 (i.e., $r = 4$) in WAS [21] and HDFS-RAID [36] in Facebook [22]. We adopt '$r = 3$' and '$r = 4$' in our experiments to resemble real-world storage cluster systems, while keeping the r value smaller than k (i.e., $r < k$).

In our experiments, the linear combination of a set of blocks is performed using the finite field arithmetic offered from the open-source Jerasure coding library [37].

The amount of an original data copy stored on each storage node is set to 8,192 MBytes (i.e., 128×64 MB), which is large enough to evaluate the archiving times of the tested solutions. The archival performance is measured in terms of time spent in archiving data of $k \times 8{,}192$ MB. We clear data buffered in memory to ensure a cold cache prior to running each experiment, which is repeatedly conducted five times to calculate the average archiving time.

We primarily focus on the 'single-disk' case in which each node is equipped with a single disk. To make a fair comparison, we incorporate both the write aggregation and shuffle mapping techniques to the DE and RapidRAID schemes. Furthermore, more often than not the read/write bandwidth of a node equipped with multiple disks is larger than that with a single disk. As such, we also investigate the five archival schemes under the 'multiple-disk' case, where each disk either stores a raw-data block or an encoded-parity block within one group. Taking node N_4 in RapidRAID as an example (see Fig. 5c), N_4 should handle two data-block reads (i.e., blocks d_1 and d_4) and one encoded-block write (i.e., block z_3). In the multiple-disk case, we place two data blocks and one encoded block on three different disks, thereby enabling each disk to service either reads or writes.

A study conducted by Lang et al. confirmed that chained declustering offers power savings [38]. To quantitatively measure the power consumed by the storage cluster running the DP and 3X schemes and their corresponding power-efficient archival schemes (i.e., $\mathrm{DP}_{\mathrm{power}}$ and $3\mathrm{X}_{\mathrm{power}}$), we use an electric power analyzer of model ZH-101 [39] to record the current of each storage node in a sample of 1 Hz.

5.3 Experimental Results

We examine the sensitivity of the five schemes to several performance factors, including the number of data nodes, the redundancy of erasure codes, the size of a storage request unit, and the number of disks in each node.

5.3.1 *k-Number of Data Nodes*

To constitute a ($k + r$, k) erasure-coded archival storage, we generate r parity blocks from k essential data blocks. We evaluate the impacts of the number k of data blocks on archival performance by setting k to 6 and 9, respectively. Fig. 7 plots the archiving times of the five schemes when SRU and r are set to 64 and 3 KB, respectively.

We observe from Fig. 7 that with the increasing value of k, the SynE scheme exhibits poorer archival performance. Such a poor performance lies in the fact that a large k value makes an excessive number of data blocks loaded to generate $r = 3$ parity blocks, which in turn leads to a long receiving time experienced by the encoding node.

DE($l = k$) exhibits long archiving time when parameter k is large, and the reason is two-fold. First, one of the r encoding nodes should read at least $\lceil k/r \rceil$ blocks, implying that the encoding node tends to process more read requests with the increasing value of k. Second, write bandwidth is reduced when an increased number of read requests compete against one write request for disk bandwidth.

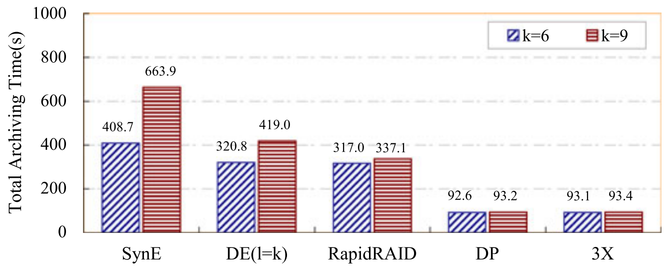


Fig. 7. Total archiving times of the five archival schemes with respect to parameter k ($k = 6$ and 9), with *SRU* = 64 KB, and $r = 3$.

In RapidRAID, k has a slim impact on the archival performance, because $k + 3$ nodes constitute an archiving pipeline and the $k - 3$ slow nodes $\{N_4, N_5, \ldots, N_k\}$ undertake an equal number of disk I/Os regardless of the k value. The $k - 3$ nodes are slow due to the following two reasons. First, the write bandwidth of these $k - 3$ nodes is lower than that of other nodes due to the bandwidth competition. Second, the write bandwidth restricts the overall archival performance (see analysis in Section 4.2).

Similar to RapidRAID, the DP and 3X schemes are slightly sensitive to parameter k in terms of archival performance. Either in DP or in 3X, all involved nodes deal with equal disk and network load when k is set to 6 and 9. Fig. 7 shows that when the k value is 6, DP outperforms SynE, DE ($l = k$), and RapidRAID in terms of archiving time by a factor of 4.41, 3.46, and 3.42, respectively; 3X surpasses SynE, DE($l = k$), and RapidRAID in terms of archiving time by a factor of 4.39, 3.45, and 3.41, respectively.

Fig. 8 shows the network traffic and the archiving times of SynE, DE($l = k$), RapidRAID, DP, and 3X in the '$k = 6$, $r = 3$' case. We observe that the trend of the network traffic is dissimilar to that of the archiving time, implying that the traffic induced by the archiving process is not a singular crucial factor affecting archival performance.

To assess computation load caused by encoding operations, we constantly monitor the CPU utilization of nodes in the storage cluster governed by the five archival schemes. Fig. 9 illustrates the average CPU utilization of nine (i.e., k + r = 9) nodes $\{N_1, N_2, \ldots, N_9\}$ under the evaluated archival schemes (i.e., SynE, DE(l = k), RapidRAID, DP, and 3X) when SRU, k, and r are set to 64, 6, and 3 KB, respectively.

We draw the following three observations from Fig. 9:

- The encoding nodes in RapidRAID exhibit the lowest computation overhead. The reason is two-fold: 1) encoding load is decentralized among all nine nodes during the entire archival process; 2) the write bandwidth restricts the archival speed, thereby resulting in low encoding intensity;
- For DP and 3X, the first encoding node in an encoding pipeline has lower CPU utilization than the other encoding nodes. This is because that only CPU overhead for local disk I/Os is involved to the first encoding step, while CPU cycles are required by both disk and network I/Os for the other encoding steps;
- Although the encoding nodes' CPU utilization in both DP and 3X is higher than that in the other three archival schemes, the encoding nodes' CPU utilization is considered reasonably low (i.e., around 30 percent); this means that computation overhead is not a dominating factor of archival performance in DP and 3X.

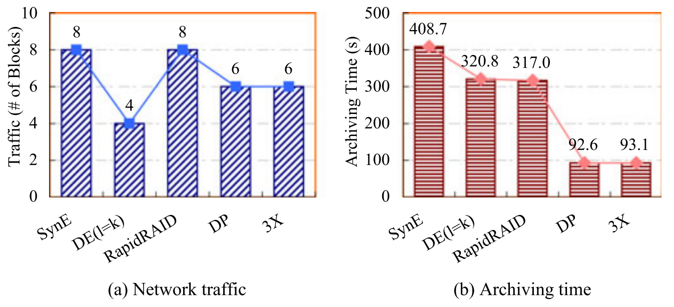


Fig. 8. Network traffic and archiving times of the SynE, DE($l = k$), RapidRAID, DP, and 3X schemes. $k = 6$ and $r = 3$.

5.3.2 r-Redundancy of Erasure Codes

In this group of experiments, we examine the sensitivity of the five archiving schemes to the redundancy r of erasure codes. We conduct the experiments on the storage cluster where the size of a request unit is 64 KB and the number k of data nodes is 6.

Fig. 10 reveals that parameter r has no noticeable impact on the archiving times of the SynE, DE($l = k$), and RapidRAID schemes. The reasons are given as follows: (1) for SynE, the network bandwidth of encoding node N_6 dominates the overall archiving performance (see Fig. 5a). In particular, the total archiving time is bounded by the receiving bandwidth of node N_6 when r is smaller than $k - 1$ (i.e., $r < k - 1$); (2) in DE($l = k$), one of r encoding nodes would deal with at least $\lceil k/r \rceil$ reads and one writes. Especially, in this group of tests, node N_7 deals with $\lceil 6/3 \rceil = 2$ reads and $\lceil 6/4 \rceil = 2$ reads when $r = 3$ and $r = 4$, respectively. This means that the same disk I/O overhead occurs in both configurations of '$k = 6, r = 3$' and '$k = 6, r = 4$'; (3) in the RapidRAID case, each of $k - r$ slow nodes $\{N_{r+1}, N_{r+2}, \ldots, N_k\}$

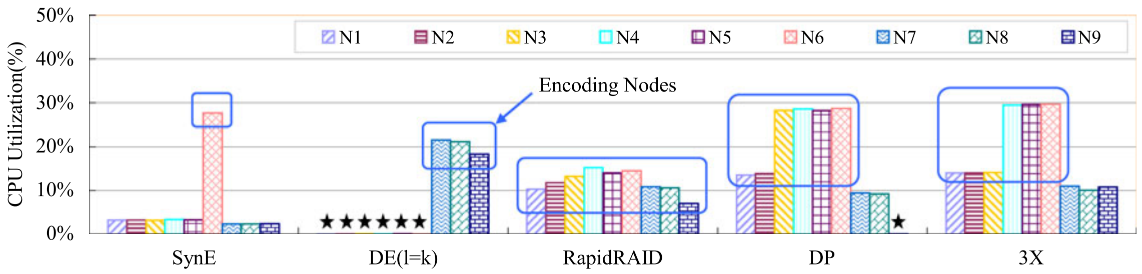


Fig. 9. Average CPU utilization of nodes $\{N_1, N_2, \ldots, N_9\}$ under the five archival schemes. Parameters k = 6, r = 3, and SRU = 64 KB. Encoding nodes are highlighted using a blue rounded rectangle; symbol ★ indicates that a node does not participate in data archival.

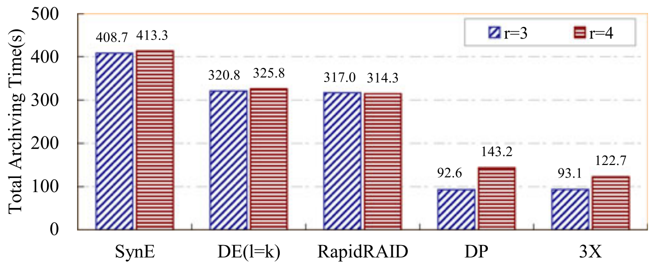


Fig. 10. Total archiving times of the five archival schemes with respect to parameter r ($r = 3$ and 4). *SRU* = 64 KB, and k = 6.

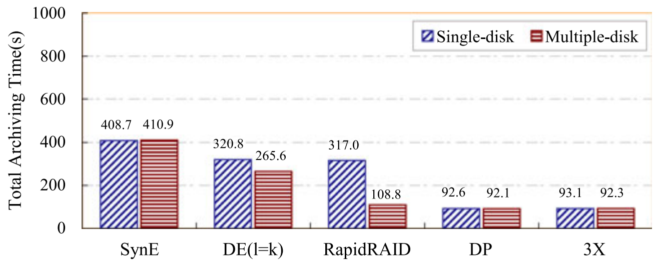


Fig. 12. Total archiving times of the five schemes in the single-disk and multiple-disk scenarios. $k = 6$, *SRU* = 64 KB, and r = 3.

responds to two reads and one write. Therefore, nodes $\{N_4, N_5, N_6\}$ for '$k = 6$, $r = 3$' and nodes $\{N_5, N_6\}$ for '$k = 6$, $r = 4$' undertake equal amount of disk I/Os, which constrain the entire archival process.

As for the DP scheme, each encoding node sends $r - 1$ intermediate parity blocks, and each of $r - 1$ nodes $\{N_{k+2}, N_{k+3}, \ldots, N_{k+r}\}$ receives two blocks. If the condition of '$r - 1 \geq 2$' holds, then the block-sending phase is still a performance bottleneck of the archival process. It is observed that when the r value is increased from 3 to 4, archiving time is enlarged by approximately 55 percent.

As to the 3X scheme, each encoding node sends r intermediate parity blocks; each of r nodes $\{N_{k+1}, N_{k+2}, \ldots, N_{k+r}\}$ receives $k/\lceil k/3 \rceil$ blocks. If the condition of '$r \geq k/\lceil k/3 \rceil$' is met, then the block-sending phase dominates the archiving performance. In particular, the block-sending phase limits the archival speed when r is greater than or equal to 3 (i.e., $r \geq 3$). The archiving-time ratio between the '$r = 4$' and '$r = 3$' cases is around 1.32, which approximates to the theoretical ratio $(r = 4)/(r = 3) \approx 1.33$.

5.3.3 SRU—Size of Storage Request Unit

To assess the impact of the request unit size or SRU, we conduct experiments on the storage cluster by setting the value of SRU to 64, 128, 256, 512, 1,024, and 2,048 KB, respectively. Fig. 11 illustrates the archiving times of the five schemes when parameters k and r are set to 6 and 3, respectively.

Fig. 11 shows that SynE, DP, and 3X are not sensitive to SRU because of two reasons. First, each node involved in the archival process responds to either reads or writes in the three schemes, and it is possible to enable the node to accomplish sequential I/Os. Second, rather than disk I/Os, it is network I/Os that dominate the overhead of the archival process.

We observe that the archival performance of the DE and RapidRAID schemes is affected by the SRU size, because there is still a proportion of non-sequential accesses even though the write aggregation and shuffle mapping techniques are incorporated. Specifically, a large SRU size helps in improving the archival performance because large SRUs boost disk I/O bandwidth by mitigating random I/O accesses. Interestingly, when the SRU size is larger than 512 KB, the archiving times of the two schemes are insensitive to the SRU size, because DE and RapidRAID gain very little benefit from saturated disk I/O bandwidth when SRUs are sufficiently large.

5.3.4 Deploying Multiple Disks

The previous sections are focused on the 'single-disk' case. Now we are positioned to evaluate the five archival schemes under the multiple-disk scenario, where a raw data block or an encoded parity block is located on a separate disk. Fig. 12 plots the archiving times of the five schemes in both the single-disk and multiple-disk cases, where k, SRU, and r are set to 6, 64, and 3 KB, respectively. Note that the archival results in the single-disk case are reported in Section 5.3.1.

From Fig. 12, we observe that regardless of SynE, DP, and 3X, each scheme exhibits similar archival performance in both cases. The reason is that it is network I/Os that dominate the archival performance of the three schemes. Take the DP scheme as an example, the network bandwidth available to two blocks is about 800 Mbps; the disk bandwidth available to two blocks is around 120 MBps in the single-disk case, and the disk bandwidth available to one block is approximately 120 MBps in the multiple-disk case (see Table 7).

The DE($l = k$) and RapidRAID schemes exhibit smaller archiving times in the multiple-disk case than in the single-disk case, because multiple disks offer higher available disk bandwidth than a single disk. It is deduced that the archival performance of DE($l = k$) is limited by disk I/Os in the single-disk case—as shown in Fig. 6b, two blocks should be sent by node N_7, then the network bandwidth available to

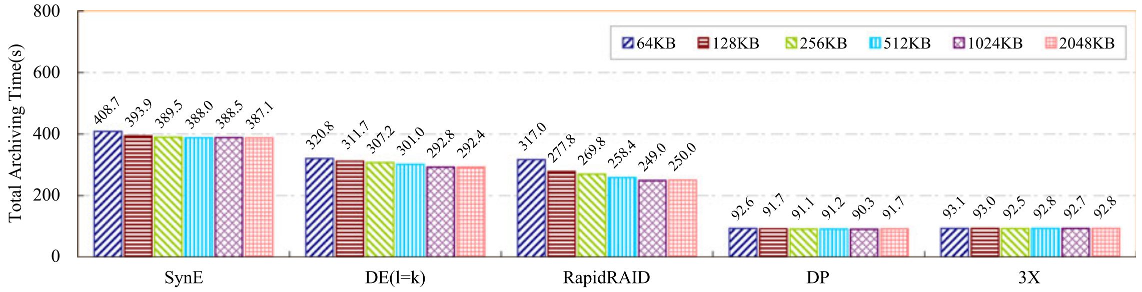


Fig. 11. Comparisons of archiving times with respect to different SRUs (64, 128, 256, 512, 1,024, and 2,048 KB). $k = 6$ and $r = 3$.

TABLE 7
Comparisons of Network and Disk Bandwidths in the Five Archival Schemes

Item Scheme		SynE	DE(l=k)	RapidRAID	DP	3X
Network bandwidth in both cases		≈800 Mbps [five blocks]	≈800 Mbps [two blocks]	≈800 Mbps [one block]	≈800 Mbps [two blocks]	≈800 Mbps [3 blocks]
Disk bandwidth	**single-disk**	≈120 MBps [one block]	≈120 MBps [three blocks]	≈120 MBps [three blocks]	≈120 MBps [two blocks]	≈120 MBps [3 blocks]
	multiple-disk	≈120 MBps [one block]	≈120 MBps [one block]	≈120 MBps [one block]	≈120 MBps [one block]	≈120 MBps [1 block]

one block would be about 400 Mbps, which is larger than the disk bandwidth available to one block (e.g., about 40 MBps). In contrast, the network I/Os restrict the archival performance of DE($l = k$) in the multiple-disk case, because the disk bandwidth available to one block increases (e.g., about 120 MBps). Furthermore, RapidRAID maximizes the utilization of both network and disk bandwidths in the multiple-disk case, thereby achieving the archival performance comparable to that of the DP and 3X schemes.

5.3.5 *Power Efficiency*

To compare the power consumption of the storage cluster governed by the tested archival schemes, we collect the power consumption of each node in the nine-node storage cluster with sample rate of 1 Hz. Fig. 13a shows archiving times under four archival scenarios (i.e., DP, $\mathrm{DP}_{\mathrm{power}}$, 3X, and $3\mathrm{X}_{\mathrm{power}}$); In Fig. 13b, we plot the average total power consumed by the storage cluster governed by each archival scheme during an entire archival process. This group of tests are conducted using parameters $k = 6$, $r = 3$, and $SRU = 64$ KB.

Three observations drawn from Fig. 13 are summarized as follows.

- The power-efficient archival schemes (i.e., $\mathrm{DP}_{\mathrm{power}}$, and $3\mathrm{X}_{\mathrm{power}}$) offer power savings at the cost of degraded archival performance. The tradeoff between archival performance and power efficiency is reasonable, because reducing the number of encoding pipelines allows some encoding nodes to be transitioned into the low-power mode, which in turn results in low archival parallelism;
- The power efficiency of the DP scheme is higher than that of the 3X scheme. Specifically, the power consumption in DP is lower than that in 3X; both the DP and 3X schemes complete the entire archival process within a similar duration. The reason lies in the fact that only eight nodes are involved in the archival process in DP, whereas there are nine nodes contributing to the archival process in 3X;
- The $\mathrm{DP}_{\mathrm{power}}$ scheme outperforms the $3\mathrm{X}_{\mathrm{power}}$ scheme. On one hand, the archiving time of $\mathrm{DP}_{\mathrm{power}}$ is smaller than that of $3\mathrm{X}_{\mathrm{power}}$ because the encoding nodes in $\mathrm{DP}_{\mathrm{power}}$ and $3\mathrm{X}_{\mathrm{power}}$ transfer two and three intermediate blocks, respectively; on the other hand, the $\mathrm{DP}_{\mathrm{power}}$ and $3\mathrm{X}_{\mathrm{power}}$ schemes consume similar energy since the same number of nodes (i.e. five nodes) are involved in the two schemes.

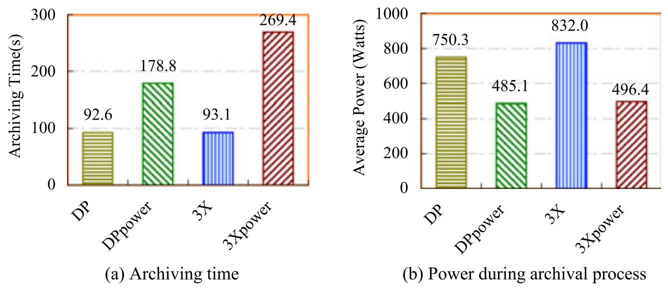


Fig. 13. Comparisons of average archiving times and average power consumption under the four archival scenarios (i.e., DP, $\mathrm{DP}_{\mathrm{power}}$, 3X, and $3\mathrm{X}_{\mathrm{power}}$). $k = 6$, $r = 3$, and $SRU = 64$ KB.

6 Related Work

Erasure-coded storage has increasingly become a cost-effective and fault-tolerant solution for archive storage systems [24], [40], [22], [41], [42]; most of erasure-coded archive storage systems are constructed upon TCP/IP-based storage clusters. For example, Pergamum is an energy-efficient disk-based archival storage [24]; Cleversafe is a cost-effective storage system for active archive storage [42]; Tahoe-LAFS is a decentralized storage system with provider-independent security for long-term storage [41].

Reed Solomon codes and their variants are widely adopted in archival storage to provide high reliability. For example, Facebook manages rarely-accessed files by deploying an open source HDFS Module called HDFS RAID, which relies on RS codes [22]; Google's GFS II adopts RS codes to archive infrequently-accessed data [7], and Windows Azure Storage (WAS) adopts a variant of RS codes (i.e., local reconstruction codes, LRC) to implement a four-fault-tolerant cluster system [21].

Most archival storage systems are designed to store newly created data using erasure codes. It is worth noting that the study of migrating existing data replicas into erasure-coded archival is rather rare. Among all the related methods found in the literature, synchronous encoding, DE and RapidRAID are the most relevant solutions to our approaches. The synchronous encoding approach creates parity blocks from three-replica redundancy using classical erasure codes [5]. Pamies-Juarez et al. proposed two solutions, namely, DE for efficient data archival [8] and RapidRAID for fast data archival [9].

In *synchronous encoding*, erasure encoding operations are conducted by a single node and the parity generation process does not exploit the three existing data replicas.

In the DE scheme, the redundancy-generation process is completed by a subset of r nodes, which act as the storing location of the encoded parity blocks. Thus, the existing k

nodes holding data blocks can be deactivated to conserve energy. Although redundancy generation traffic in DE can be significantly reduced, the traffic does not determine the overall archiving time. Furthermore, the fault tolerance is a function of the number l of blocks stored in the r coding nodes; the DE codes cannot guarantee the MDS feature unless the value of l is $2k$.

RapidRAID is a family of erasure codes that realize the idea of pipelined erasure coding to speed up the data archival process. RapidRAID codes are non-systematic, meaning that it is required to decode the archived data to service any read request, thereby introducing data-access overhead. The $(k+r, k)$ RapidRAID codes might have a fault tolerance problem if the redundancy parameter r is larger than three (i.e., $r > 3$).

Different from RapidRAID, our DP and 3X schemes migrate data replicas to encoded blocks using the Reed Solomon codes. Both DP and 3X allow fore-end users to access archived data without imposing decoding operations. Contrary to DE and RapidRAID that suffer from the problem of undetermined fault tolerance, both DP and 3X are able to guarantee the MDS property for all coding parameters k and r. With layouts $[\mathrm{D+P}]_{\mathrm{cd}}$ and $[3\mathrm{X}]_{\mathrm{cd}}$ in place, the DP and 3X schemes allow $\lceil (k+1)/2 \rceil$ and k$-\lceil k/3 \rceil$ nodes to be transitioned into the low-power mode while keeping other nodes active to perform the archival process, respectively.

7 Further Discussions

Although $[\mathrm{D+P}]_{\mathrm{cd}}$ and $[3\mathrm{X}]_{\mathrm{cd}}$ are capable of tolerating double failures, the storage overhead of the $[\mathrm{D+P}]_{\mathrm{cd}}$ layout is smaller than that of $[3\mathrm{X}]_{\mathrm{cd}}$. Specifically, $[3\mathrm{X}]_{\mathrm{cd}}$ suffers from 200 percent storage overhead since it replicates three copies for each block [5], whereas the storage overhead of $[\mathrm{D+P}]_{\mathrm{cd}}$ is reduced down to $1 + 2/k$.

If k (for, $k \geq 3$) is not a multiple of three in the 3X archival scheme, there are just two archiving pipelines, which deliver approximately 2/3 of the archival performance of the case of '$k\%3 = 0$'. In most production clusters, the k value is a multiple of three; for example, k is set to six in both GFS II [7] and QFS [28], k equals to 9 in HYDRAstor [35], and k is configured to 12 in WAS [6].

It is supposedly true that the performance bottleneck (i.e., block-receiving stage) of SynE is eliminated by the 10 Gbps-Ethernet network of a storage cluster. However, a majority of data centers (e.g., Google [7], WAS [6], and Facebook [22]) are employing 1-Gbps Ethernet due to its high cost-effectiveness. Therefore, it is indispensable to take the GbE interconnect into account when one designs archival schemes in the context of storage clusters.

It is unnecessary to deploy the 'shuffle mapping' solution to the DE($l = k$), RapidRAID, DP, and 3X schemes to improve the I/O sequentiality in the multi-disk case, where each requested block within a stripe is residing on an individual disk, naturally guaranteeing high disk I/O bandwidth.

We have investigated the DE scheme in the case of '$l = k$'. The DE($l = 2k$) scheme is expected to have lower archival performance than that of DE($l = k$), because the write bandwidth in the '$l = 2k$' case is suppressed by the disk bandwidth competition between an increased number of reads and one write.

8 Conclusion and Future Work

In this paper, we proposed two pipelined archival schemes called *DP* and *3X* to boost the erasure-coded archival performance on storage clusters. We incorporated the chained-declustering mechanism into both *Mirrored RAID-5* and *triplication* redundancy groups, resulting in data layouts $[\mathrm{D+P}]_{\mathrm{cd}}$ and $[3\mathrm{X}]_{\mathrm{cd}}$. These two data layouts enable a subset of involved nodes to constitute an encoding pipeline, and they allow two or three encoding pipelines to generate parity blocks in parallel. We implement both the *DP* and *3X* schemes and three existing solutions (i.e., SynE, DE, and RapidRAID) in a real-world storage cluster. Extensive experimental results show that our archival schemes respectively outperforms SynE, DE($l = k$), and RapidRAID by a factor of at least 4.39, 3.45, and 3.41 in a nine-node storage cluster. We draw two interesting observations in this study. First, the performance bottleneck of SynE lies in its block-receiving stage; the dominant factor of archival performance is disk I/Os rather than network I/Os in both DE and RapidRAID. Second, there exists a tradeoff between energy efficiency and archival performance in the two pipelined archival schemes, where the performance-oriented schemes exhibit higher archival performance than the energy-efficient archival schemes.

In this study, we paid particular attention to the off-line archival process—no user I/Os are issued and raw data blocks are retrieved from disks. As a future research direction, we plan to investigate an online archival process, in which we intend to optimize archival performance by (1) exploiting the existence of hot user data and (2) caching the hot data in an in-memory system.

Acknowledgments

This work is supported in part by the National High Technology Research Program of China under Grant No. 2013AA013203, the National Basic Research Program of China under Grant No. 2011CB302303, and the Fundamental Research Funds for the Central Universities under No. 2014QN012. Xiao Qin's work is supported by the U.S. National Science Foundation under Grants CCF-0845257 (CAREER), CNS-0917137 (CSR), CNS-0757778 (CSR), CNS-0831502 (CyberTrust), and CCF-0742187 (CPA). Shu Yin's work is supported by the National Natural Science Foundation of China under Grant NO.61402158. The authors also appreciate Zhongyuan Min for his assistance in the power evaluation for this study. Changsheng Xie is the corresponding author of this paper.

References

[1] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," *ACM SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 29–43, 2003.
[2] D. Borthakur. (2009). The Hadoop distributed file system: Architecture and design, Hadoop Apache Project [Online]. Available: http://hadoop.apache.org/common/docs/current/hdfs_design.html
[3] Amazon-INC. (2010). Amazon simple storage services (Amazon S3) [Online]. Available: http://aws.amazon.com/s3
[4] H. Weatherspoon, and J. D. Kubiatowicz, "Erasure coding vs. replication: A quantitative comparison," in *Proc. 1st Int. Workshop Peer-to-Peer Syst.*, 2002, pp. 328–337.

[5] B. Fan, W. Tantisiriroj, L. Xiao, and G. Gibson, “Diskreduce: Replication as a prelude to erasure coding in data-intensive scalable computing,” *Proc. Int. Conf. High Perform. Comput. Netw., Storage Anal.*, 2011, pp. 6–10.

[6] B. Calder, J. Wang, A. Ogus, N. Nilakantan, A. Skjolsvold, S. McKelvie, Y. Xu, S. Srivastav, J. Wu, H. Simitci, J, Haridas, C. Uddaraju, H. Khatri, A. Edwards, V. Bedekar, S. Mainali, R. Abbasi, A. Agarwal, M. F. ul Haq, M. I. ul Haq, D. Bhardwaj, S. Dayanand, A. Adusumilli, M. McNett, S. Sankaran, K. Manivannan, and L. Rigas, “Windows azure storage: A highly available cloud storage service with strong consistency,” in *Proc. 23rd ACM Symp. Oper. Syst. Principles*, 2011, pp. 143–157.

[7] D. Ford, F. Labelle, F. Popovici, M. Stokely, V. Truong, L. Barroso, C. Grimes, and S. Quinlan, “Availability in globally distributed storage systems,” in *Proc. 9th USENIX Symp. Oper. Syst. Des. Implementation*, 2010, pp. 61–74.

[8] L. Pamies-Juarez, F. Oggier, and A. Datta, “Decentralized erasure coding for efficient data archival in distributed storage systems,” in *Distributed Computing and Networking*. New York, NY, USA: Springer, 2013, pp. 42–56.

[9] L. Pamies-Juarez, A. Datta, and F. Oggier, “RapidRAID: Pipelined erasure codes for fast data archival in distributed storage systems,” in *Proc. 32nd IEEE Int. Conf. Comput. Commun.*, 2013, pp. 1294–1302.

[10] H.-I. Hsiao, and D. J. DeWitt, “Chained declustering: A new availability strategy for multiprocessor database machines,” in *Proc. 6th Int. Conf. Data Eng.*, 1990, pp. 456–465.

[11] S. Chen, and D. Towsley, “A performance evaluation of RAID architectures,” *IEEE Trans. Comput.*, vol. 45, no. 10, pp. 1116–1130, Oct. 1996.

[12] H. Zhu, P. Gu, and J. Wang, “Shifted declustering: A placement-ideal layout scheme for multi-way replication storage architecture,” in *Proc. 22nd Annu. Int. Conf. Supercomput.*, 2008, pp. 134–144.

[13] Q. Xin, E. L. Miller, T. Schwarz, D. D. Long, S. A. Brandt, and W. Litwin, “Reliability mechanisms for very large storage systems,” in *Proc. 20th IEEE/11th NASA Goddard Conf. Mass Storage Syst. Technol.*, 2003, pp. 146–156.

[14] K. M. Greenan, E. L. Miller, T. J. Schwarz, and D. D. Long, “Disaster recovery codes: Increasing reliability with large-stripe erasure correcting codes,” in *Proc. ACM Workshop Storage Security Survivability*, 2007, pp. 31–36.

[15] B. Fan, W. Tantisiriroj, L. Xiao, and G. Gibson, “DiskReduce: RAID for data-intensive scalable computing,” in *Proc. 4th Annu. Workshop Petascale Data Storage*, 2009, pp. 6–10.

[16] D. Borthakur, “The Hadoop distributed file system: Architecture and design,” *Hadoop Project Website*, vol. 11, p. 21, 2007.

[17] K. M. Greenan, D. D. Long, E. L. Miller, S. Schwarz, and J. J. Wylie, “A spin-up saved is energy earned: Achieving power-efficient, erasure-coded storage,” in *Proc. 4th Conf. Hot Topics Syst. Dependability*, 2008, pp. 4–9.

[18] J. Huang, F. Zhang, X. Qin, and C. Xie, “Exploiting redundancies and deferred writes to conserve energy in erasure-coded storage clusters,” *ACM Trans. Storage*, vol. 9, no. 2, p. 4, 2013.

[19] J. Plank, “A Tutorial on Reed-Solomon Coding for Fault-tolerance in RAID-like Systems,” *Softw. Pract. Exp.*, vol. 27, no. 9, pp. 995–1012, 1997.

[20] M. Manasse, C. Thekkath, and A. Silverberg, “A reed-solomon code for disk storage, and efficient recovery computations for erasure-coded disk storage,” *Proc. Inform.*, 2009, pp. 1–11.

[21] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin, “Erasure coding in windows azure storage,” in *Proc. USENIX Conf. Annu. Tech. Conf.*, 2012, pp. 15–26.

[22] A. Thusoo, Z. Shao, S. Anthony, D. Borthakur, N. Jain, J. Sarma, R. Murthy, and H. Liu, “Data warehousing and analytics infrastructure at Facebook,” in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2010, pp. 1013–1020.

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

[24] M. Storer, K. Greenan, E. Miller, and K. Voruganti, “Pergamum: Replacing tape with energy efficient, reliable, disk-based archival storage,” in *Proc. 6th USENIX Conf. File Storage Technol.*, 2008, pp. 1–16.

[25] J. Plank, J. Luo, C. Schuman, L. Xu, and Z. Wilcox-O’Hearn, “A performance evaluation and examination of open-source erasure coding libraries for storage,” in *Proc. 7th Conf. File Storage Technol.*, 2009, pp. 253–265.

[26] T. Karslsson and L. Lundberg, “Performance evaluation of cauchy reed-solomon coding on multicore systems,” in *Proc. IEEE 7th Int. Symp. Embedded Multicore Socs*, 2013, pp. 165–170.

[27] J. S. Plank, M. Blaum, and J. L. Hafner, “SD codes: Erasure codes designed for how storage systems really fail,” in *Proc. 11th USENIX Conf. File Storage Technol.*, 2013, pp. 95–104.

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

[29] Q. Yang and Y. Hu, “DCD—Disk caching disk: A new approach for boosting I/O performance,” in *Proc. 23rd Annu. Int. Symp. Comput. Archit.*, 1996, pp. 169–181.

[30] J. Wang and Y. Hu, “WOLF-A novel reordering write buffer to boost the performance of log-structured file systems,” in *Proc. 1st USENIX Conf. File Storage Technol.*, 2002, pp. 47–60.

[31] J. Huang, X. Liang, X. Qin, P. Xie, and C. Xie, “Scale-RS: An efficient scaling scheme for rs-coded storage clusters,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 99, no. 1, p. 1, PrePrints, doi:10.1109/TPDS.2014.2326156, 2014.

[32] S. B. Morris and R. E. Hartwig, “The generalized faro shuffle,” *Discrete Math.*, vol. 15, no. 4, pp. 333–346, 1976.

[33] T. IOMETER. (1997). IOMETER: I/O subsystem measurement and characterization tool open source code distribution [Online]. Available: http://www.iometer.org

[34] S. Sumimoto, K. Ooe, K. Kumon, T. Boku, M. Sato, and A. Ukawa, “A scalable communication layer for multi-dimensional hyper crossbar network using multiple gigabit ethernet,” in *Proc. 20th Int. Conf. Supercomput.*, 2006, pp. 107–115.

[35] C. Ungureanu, B. Atkin, A. Aranya, S. Gokhale, S. Rago, G. Calkowski, C. Dubnicki, and A. Bohra, “HydraFS: A high-throughput file system for the HYDRAstor content-addressable storage system,” in *Proc. 8th USENIX Conf. File Storage Technol.*, 2010, pp. 225–238.

[36] D. Borthakur, R. Schmidt, R. Vadali, S. Chen, and P. Kling, “HDFS RAID,” technical Talk. Yahoo! Developer Network, 2010.

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

[38] W. Lang, J. M. Patel, and J. F. Naughton, “On energy management, load balancing and replication,” *ACM SIGMOD Rec.*, vol. 38, no. 4, pp. 35–42, 2010.

[39] Zyhd. (2010). ZH-101 portable electric power fault recorder and analyzer [Online]. Available: http://www.zyhd.com.cn/cN/Bs_Product.asp

[40] S. Frolund, A. Merchant, Y. Saito, S. Spence, and A. Veitch, “A decentralized algorithm for erasure-coded virtual disks,” in *Proc. Int. Conf. Dependable Syst. Netw.*, 2004, pp. 125–134.

[41] Z. Wilcox-O’Hearn and B. Warner, “Tahoe: The least-authority filesystem,” in *Proc. 4th ACM Int. Workshop Storage Security Survivability*, 2008, pp. 21–26.

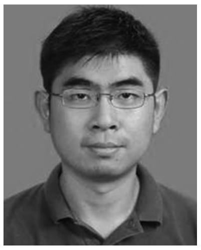
[42] I. CLEVERSAFE. (2008). Cleversafe dispersed storage [Online]. Available: open source code distribution: http://www.cleversafe.org/downloads

Jianzhong Huang received the PhD degree in computer architecture in 2005 and the postdoctoral research in information engineering in 2007 from the Huazhong University of Science and Technology (HUST), Wuhan, China. He is currently an associate professor in the Wuhan National Laboratory for Optoelectronics at HUST. His research interests include computer architecture and dependable storage systems. He received the National Science Foundation of China in Storage System Research Award in 2007. He is a member of the China Computer Federation (CCF).

Yanqun Wang received the BS degree in computer science and technology in 2013 from the Changsha University of Science and Technology, China. She is currently working toward the MS degree at the Huazhong University of Science and Technology. Her research interests include clustered storage and erasure-coded data archival.

Xiao Qin (S'00-M'04-SM'09) received the BS and MS degrees in computer science from the Huazhong University of Science and Technology, Wuhan, China, and the PhD degree in computer science from the University of Nebraska-Lincoln, in 1992, 1999, and 2004, respectively. He is currently an associate professor with the Department of Computer Science and Software Engineering, Auburn University. His research interests include parallel and distributed systems, storage systems, fault tolerance, real-time systems, and performance evaluation. He received the US National Science Foundation (NSF) Computing Processes and Artifacts Award and the NSF Computer System Research Award in 2007, and the NSF CAREER Award in 2009. He is a senior member of the IEEE.

Xianhai Liang received the BS degree in computer science and technology in 2012 from the Wuhan University of Technology, China. He is currently working toward the MS degree at the Huazhong University of Science and Technology. His research interests include networked storage systems and file system.



Shu Yin received the BS and MS degrees in communication engineering from the Wuhan University of Technology, China, and the PhD degree in computer science from Auburn University, in 2006, 2008, and 2012, respectively. He is currently an assistant professor in Hunan University, China. His research interests include storage systems, reliability modeling, fault tolerance, energy-efficient computing, high-performance computing, and wireless communications.

Changsheng Xie received the BS and MS degrees in computer science both from the Huazhong University of Science and Technology (HUST), Wuhan, China, in 1982 and 1988, respectively. He is currently a professor in the Department of Computer Engineering at HUST. He is also the director of the Data Storage Systems Laboratory of HUST and the deputy director of the Wuhan National Laboratory for Optoelectronics. His research interests include computer architecture, I/O system, and networked storage system. He is the vice chair of the expert committee of Storage Networking Industry Association (SNIA), China. He is a member of the IEEE.