IEICE TRANS. INF. & SYST., VOL.E100–D, NO.2 FEBRUARY 2017

PAPER

Fast Reconstruction for Degraded Reads and Recovery Process in Primary Array Storage Systems*

Baegjae SUNG[†a)], ***Nonmember and*** **Chanik PARK**[†b)], ***Member***

SUMMARY RAID has been widely deployed in disk array storage systems to manage both performance and reliability simultaneously. RAID conducts two performance-critical operations during disk failures known as *degraded reads/writes* and *recovery process*. Before the recovery process is complete, reads and writes are degraded because data is reconstructed using data redundancy. The performance of degraded reads/writes is critical in order to meet stipulations in customer service level agreements (SLAs), and the recovery process affects the reliability of a storage system considerably. Both operations require fast data reconstruction. Among the erasure codes for fast reconstruction, Local Reconstruction Codes (LRC) are known to offer the best (or optimal) trade-off between storage overhead, fault tolerance, and the number of disks involved in reconstruction. Originally, LRC was designed for fast reconstruction in distributed cloud storage systems, in which network traffic is a major bottleneck during reconstruction. Thus, LRC focuses on reducing the number of disks involved in data reconstruction, which reduces network traffic. However, we observe that when LRC is applied to primary array storage systems, a major bottleneck in reconstruction results from *uneven disk utilization*. In other words, underutilized disks can no longer receive I/O requests as a result of the bottleneck of overloaded disks. Uneven disk utilization in LRC is due to its *dedicated group partitioning* policy to achieve the *Maximally Recoverable* property. In this paper, we present Distributed Reconstruction Codes (DRC) that support fast reconstruction in primary array storage systems. DRC is designed with *group shuffling* policy to solve the problem of uneven disk utilization. Experiments on real-world workloads show that DRC using global parity rotation (DRC-G) improves degraded performance by as much as 72% compared to RAID-6 and by as much as 35% compared to LRC under the same reliability. In addition, our study shows that DRC-G reduces the recovery process completion time by as much as 52% compared to LRC.

key words: *array storage systems, RAID, erasure codes, fast reconstruction*

1. Introduction

The data protection technique of RAID [1] has been widely deployed in primary disk array storage systems to manage both performance and reliability simultaneously. RAID recovers data when a disk failure occurs by using redundant data (e.g., erasure codes). The two performance-critical operations of RAID during disk failures are known as *degraded reads/writes* and *recovery process*. If a disk failure occurs, RAID starts the recovery process. The background recovery process reconstructs data of the failed disk and rebuilds data onto a replacement disk. Simultaneously, RAID serves reads and writes from applications using data reconstruction (i.e., degraded reads/writes). Therefore, fast reconstruction is the main operation that improves both the performance of degraded reads/writes and the recovery process.

In primary array storage systems, the performance of degraded reads/writes during disk failure is critical in order to meet stipulations in customer service level agreements (SLAs). In addition, the performance of the recovery process affects the period between fault and complete recovery, thus affecting the reliability of storage systems.

Two approaches are used to support fast reconstruction. The first approach [2]–[6] involves reducing the amount of data read from disks for reconstruction. This approach focuses on finding coding coefficients for reconstruction or optimizing a reconstruction algorithm while maintaining the *Maximum Distance Separable* (MDS) property. Therefore, this approach is known to achieve lower reconstruction performance than do non-MDS codes. The second approach [7]–[11] involves reducing the number of disks for reconstruction. This approach focuses on constructing efficient erasure codes necessary for the trade-off between storage overhead, fault tolerance, and the number of disks involved in reconstruction under non-MDS. Among these codes, Local Reconstruction Codes (LRC) [10], [11] are known to offer the best (or optimal) trade-off.

Originally, LRC was designed for distributed cloud storage service. Because network traffic is the cause of a major bottleneck in the distributed environment, LRC focuses on reducing the number of disks involved in reconstruction in order to reduce network traffic. LRC has now been widely applied to primary array storage systems [13], [14]. However, we have observed that, when LRC is applied to primary array storage systems, a major bottleneck in reconstruction results from *uneven disk utilization*. In other words, underutilized disks can no longer receive I/O requests as a result of the bottleneck of overloaded disks.

In this paper, we present Distributed Reconstruction Codes (DRC) to support fast reconstruction in primary array storage systems. DRC uses a *group shuffling* policy to solve uneven disk utilization without considerably sacrificing reliability. The contributions of this paper are as follows:

1. We identified the reason for low reconstruction performance on LRC in primary array storage systems: un-

Manuscript received May 19, 2016.
Manuscript revised September 13, 2016.
Manuscript publicized November 11, 2016.

[†]The authors are with Dept. of Computer Science and Engineering, POSTECH, 37673 Republic of Korea.

[*]This study was supported by Brain Korea 21 PLUS project for POSTECH Computer Science & Engineering Institute.

a) E-mail: jays@postech.ac.kr
b) E-mail: cipark@postech.ac.kr

DOI: 10.1587/transinf.2016EDP7208

even disk utilization.

2. We propose DRC, which supports fast reconstruction in primary array storage systems.
3. We analyze the trade-off between degraded read performance and the reliability of DRC.

This paper is organized as follows. Section 2 identifies the reason for low reconstruction performance in array storage systems when using erasure codes, RAID-6 and LRC. Section 3 explains the design of our DRC and analyzes degraded read performance and reliability. Section 4 presents an experimental evaluation. Section 5 lists related work on erasure codes. Section 6 concludes our study.

2. Backgrounds and Motivation

2.1 RAID-6

RAID-6 uses multiple disks that are grouped together to improve reliability and performance. The improvement is achieved through the concepts of data striping and data redundancy. Data striping places different parts of data on different disks. Each portion is often referred to as *stripe*, whereas the parts are termed *stripe units* or *chunks*. The two sets of redundancy data (or parity), which we call P parity and Q parity, ensure all data are recoverable when no more than two disk failures occur. The P parity is generated by means of XOR-summing all data chunks, and the Q parity is generated by means of Reed-Solomon (RS) coding. Figure 1(a) provides a simple example of RAID-6 (4, 2) in which four disks are used to store data chunks and two disks are used to store parity chunks. Although for simplicity we describe the examples without parity rotation, we assume that multiple stripes exist and that all chunks are rotated between stripes. Thus, all disks will have the same workload.

Figure 1(b) shows the manner in which to reconstruct missing data chunk d_0 resulting from a failure of disk #0. RAID reads data chunks d_1, d_2, d_3 and parity chunk p_0, and then decodes the missing data chunk d_0. The number of chunks required to read during reconstruction is usually called *reconstruction cost*. The reconstruction is performed based on two conditions. First, the missing data chunk is required in order to operate read/write requests from applications. Second, the missing data chunk is required to rebuild the replacement disk by means of the recovery process.

Let us now consider degraded read performance. Degraded reads occur a mixture of *direct read* (reading from a surviving data chunk) and *reconstruction read* (reading from a missing data chunk). On average, the same number of read requests are then sent to each disk that stores data chunks. We herein define a data disk as a disk that stores data chunks. Figure 1(c) gives an example of read operations on degraded reads, and we assume that each data disk receives a single read request. Each of the surviving data disks #1, #2, and #3 performs two read operations: one for a direct read and one for a reconstruction. Thus, we observe that reconstruction generates additional read operations that cause performance degradation. In summary, reconstruction cost is 4, and each surviving data disk performs x2 read operations.

2.2 LRC

LRC [10], [11] was originally proposed for fast reconstruction in distributed cloud storage systems. A configuration of LRC is defined by three attributes (k, l, r), where k is the number of disks used to store data chunks, l is the number of disks used to store local parity chunks, and r is the number of disks used to store global parity chunks. Note that $k + l + r$ equals the total number of disks. Figure 2(a) provides a simple example of LRC (4, 2, 1). Data chunks d_0 and d_1 are protected by parity chunk p_0, and data chunks d_2 and d_3 are protected by parity chunk p_1. These two parity groups are called a *local group*. In addition, all data chunks d_0, d_1, d_2, and d_3 are protected by parity chunk q_0. This parity group is called a *global group*.

Local parity groups play a major role in reducing reconstruction cost. For example, in Fig. 2(b), let us assume that data chunk d_0 is lost. To reconstruct data chunk d_0, reading data chunk d_1 and parity chunk p_0 all within the local parity group is sufficient. Its reconstruction cost is 2, which is much smaller than that of RAID-6.

Figure 2(c) provides an example of read operations for degraded reads. Unlike RAID-6, every surviving data disk within a faulty local group performs x2 read operations. In Fig. 2(c), data disk #1 is the only live data disk within the faulty local group.

Regarding fault tolerance, each data chunk belongs to two parity groups, that is, to one local parity group and to

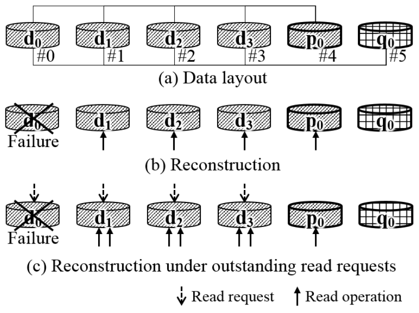

Fig. 1 Data layout and reconstruction on RAID-6 (4, 2).

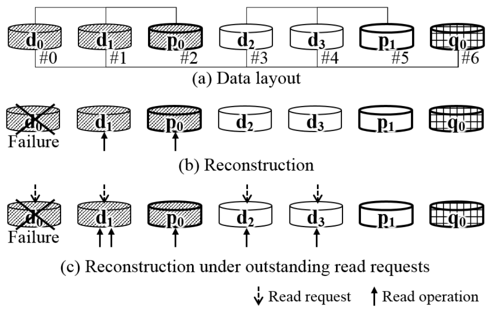

Fig. 2 Data layout and reconstruction on LRC (4, 2, 1).

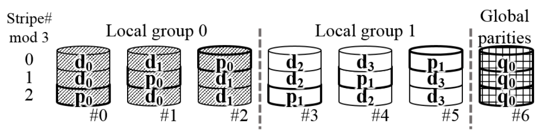

Fig. 3 Parity-rotated data layout on LRC (4, 2, 1).

one global parity group. Thus, LRC (4, 2, 1) can tolerate two arbitrary disk failures. In the event of three disk failures, LRC may recover data depending on the failed disk location. If, when recovering three disk failures, all three parities (e.g., two local and one global) are utilized, LRC can recover the data; otherwise, it cannot. For example, LRC (4, 2, 1) cannot recover data if three failed disks are #0, #1, and #2. We define p_d as the recoverability ratio between recoverable and unrecoverable cases in three disk failures. LRC (4, 2, 1) has a p_d value of 0.77. This means that LRC (4, 2, 1) can recover data in only 77% of all three disk failure cases. Thus, we can call p_d the percentage of decodability. Note that LRC builds multiple local groups and each local parity group is disjoint from each other (we call this the *dedicated group partitioning* policy). Thanks to this dedicated group partitioning policy, LRC achieves high recoverability.

LRC can control the trade-off between reconstruction cost and fault tolerance by configuring the number of local parity and global parity. For example, if we increase the number of local parity in LRC, the reconstruction cost can be reduced because each local parity group size is reduced. Simultaneously, the p_d value can be increased, resulting in increased reliability. The number of global parity affects the reliability of a storage system more considerably than the number of local parity. However, increasing the number of global parity highly degrades write performance.

2.3 Limitations of LRC for Array Storage Systems

The parity rotation technique may be easily applied to LRC to improve performance without affecting the reliability of a storage system. Figure 3 shows a parity-rotated data layout of LRC (4, 2, 1).

To investigate the extent to which LRC can be applied to a primary array storage system, we conducted an experiment to measure degraded read performance on a primary array storage system. (Refer to Sect. 4 for a more detailed description of the experimental environment employed in this study.) However, as shown in Fig. 4, the performance improvement of degraded reads in LRC compared to that in RAID-6 is insignificant: a mere 3% improvement on small reads. This result exceeds our expectations because the performance gain of LRC over that of RAID-6 (or RS) in degraded reads was reported as 29-32% in [10].

After careful analysis of the behavior of LRC in a primary array storage system, we identified the cause of the problem: uneven disk utilization. Figure 5 shows the resulting disk utilization of LRC (4, 2, 1) when it is applied to a primary array storage system. Note that the parity-rotated data layout of LRC in Fig. 3 is used in the experiment. The

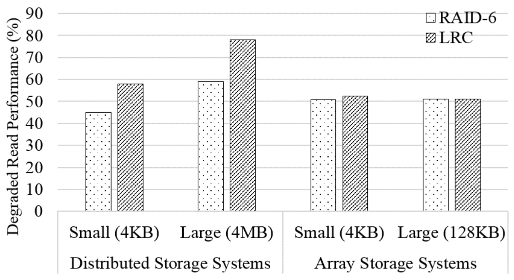

Fig. 4 Degraded read performance of RAID-6 (4, 2) and LRC (4, 2, 1) during a single disk failure. Degraded read performance is normalized to normal read performance. The results of distributed storage systems are obtained from [10], and the results of array storage systems are measured on our testbed (please refer to Sect. 4).

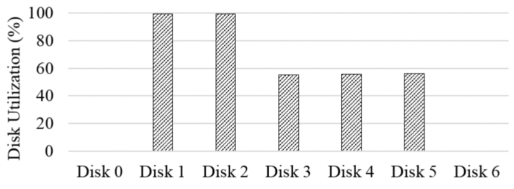

Fig. 5 Disk utilization of LRC (4, 2, 1) during a single disk failure (#0) on a sequential read (128 KB) workload.

main reason disks #3, #4, and #5 are underutilized is that underutilized disks can no longer receive I/O requests as a result of the bottleneck of overloaded disks #1 and #2. For example, in Fig. 3, assume that each of the disks from #0 to #5 receives two read requests during a failure of disk #0. Each of the disks #1 and #2 performs four read operations (i.e., two for direct reads and two for reconstructions) while each of the disks #3, #4 and #5 performs two read operations. This means that each of the overloaded disks performs x2 more read operations than each of the underutilized disks. The number of outstanding requests sent to an array storage system (or queue depth) is limited to a certain threshold (e.g., 128), and no additional I/O requests can be sent to the underutilized disks until overloaded disks complete their assigned I/O requests. Therefore, reducing performance degradation of overloaded disks is a major challenge to improving degraded read performance for array storage systems.

3. DRC

3.1 Group Shuffling and Parity Rotation

We observe that uneven disk utilization during data reconstruction in LRC is caused by the fact that local parity groups are independently organized. In other words, data disks of each local group are disjointed. To address this problem, we must develop new code in which the data disks of each local parity group are no longer disjointed.

In this paper, we present a novel erasure code DRC. The main technique we employ is called *group shuffling*. Note that, in LRC, local groups have disjointed sets of data disks, and we call this dedicated group partitioning.

Algorithm 1 Local group number decision algorithm for DRC (*k*, *l*, *r*)

```
Input the index of a data chunk $n$
    // $n$ is computed by logical block address/data chunk size //
Output local group number the input data chunk belongs to
    // $0 \sim l-1$ //
$i \leftarrow n/k$                        // stripe number of data chunk
$j \leftarrow n \bmod k$                  // index of data chunk within a stripe
$base \leftarrow j/(k/l)$                 // local group of data chunk in LRC
$pos \leftarrow j \bmod (k/l)$            // position of data chunk within local group
$degree \leftarrow i/l^{pos}$             // the amount of shuffling applied to data chunk
$group \leftarrow (base + degree) \bmod l$  // local group for data chunk
return group
```

We next describe the manner in which data layout of DRC is configured. First, similar to LRC, DRC is defined by three attributes (k, l, r), where k is the number of data chunks in a stripe, l is the number of local parity chunks in a stripe, and r is the number of global parity chunks in a stripe. That is because each stripe has a different data layout in DRC, whereas every stripe has the same data layout in LRC. We assume that k is divisible by l.

Given DRC (k, l, r), we must determine data layout patterns in which k disks are equally involved in data reconstruction. Note that, regarding LRC, only k/l disks are involved in data reconstruction. Algorithm 1 shows the method used to decide which local parity group a given data chunk belongs to. Algorithm 1 is invoked whenever write requests are issued to disks or read requests are issued to faulty disks. For each request, Algorithm 1 decides which local group handles the request, and which disks are used to store data chunks inside the local group.

Theorem 1. *The data layouts of DRC (k, l, r) generated by Algorithm 1 have the following properties:*

1. *data layout patterns repeat every $l^{(k/l)}$ stripes.*
2. *k disks of all local groups are equally involved in data reconstruction.*

Proof. We denote the local group number (LGN) for the given data chunk of n as $g_{i,j}$, where $i = n/k$ and $j = n \bmod k$ from Algorithm 1.

(1) Define $R_i = \langle g_{i,j} \mid 0 \le j < k \rangle$ as the sequence of LGNs for data chunks in i-th stripe. We will show that $R_i = R_{i+m\times l^{(k/l)}}$ for $m \ge 1$ and $0 \le i < l^{(k/l)}$. From the algorithm, we get $g_{i,j} = (base + \lfloor i/l^{pos} \rfloor) \bmod l$, and $g_{i+m\times l^{(k/l)},j} = (base + \lfloor i/l^{pos} + m \times l^{(k/l)}/l^{pos} \rfloor) \bmod l$. Since $0 \le pos < k/l$, $m \times l^{(k/l)}/l^{pos} \bmod l$ is always equal to 0. Hence, $g_{i+m\times l^{(k/l)},j} = g_{i,j}$ and Property 1 is satisfied.

(2) Without loss of generality, assume j-th data disk fails. Define $C_j = \langle g_{i,j} \mid 0 \le i < l^{(k/l)} \rangle$ as the sequence of LGNs for data chunks in j-th data disk. Due to repeated data layouts of Property 1, it is enough to consider $g_{i,j}$ for $0 \le i < l^{(k/l)}$. Note that, in order to reconstruct $l^{(k/l)}$ data chunks of j-th data disk, $l^{(k/l)}$ local parity chunks and $(k/l - 1) \times l^{(k/l)}$ data chunks are read. So, we will show that $l^{(k/l)}$ local parity chunks and $(k/l - 1) \times l^{(k/l)}$ data chunks are equally distributed in l local parity disks and $k - l$ data disks, respectively.
First, we show that, for each local group p $(0 \le p < l)$, C_j has $l^{(k/l-1)}$ LGNs satisfying $g_{i,j} = p$. Since $g_{i,j} = (base + \lfloor i/l^{pos} \rfloor) \bmod l$, $g_{i,j} = p \Leftrightarrow \lfloor i/l^{pos} \rfloor \equiv p - base \pmod{l}$. Let $x = \lfloor i/l^{pos} \rfloor$. Because x is $0 \le x < l^{(k/l-pos)}$, the number of x satisfying $x \equiv p - base \pmod{l}$ is $l^{(k/l-pos-1)}$. Each value of x represents l^{pos} LGNs, thus there are $l^{(k/l-1)}$ LGNs satisfying $g_{i,j} = p$. This means that each of l local parity disks stores $l^{(k/l-1)}$ local parity chunks for reconstructing $l^{(k/l)}$ data chunks of j-th data disk.
Second, we show that, for each working data disk j' $(0 \le j' < k$ and $j' \not\equiv j \pmod{k/l})$, C_j has $l^{(k/l-1)}$ LGNs satisfying $g_{i,j} = g_{i,j'}$. Since $g_{i,j} = (base + \lfloor i/l^{pos} \rfloor) \bmod l$, $g_{i,j} = g_{i,j'} \Leftrightarrow \lfloor i/l^{pos} \rfloor \equiv \lfloor i/l^{pos'} \rfloor + base' - base \pmod{l}$. Let $x = \lfloor i/l^{pos} \rfloor$ and $x' = \lfloor i/l^{pos'} \rfloor$. Case 1) $pos > pos'$: The values of $x' \bmod l$ are equally distributed values from 0 to $l - 1$ while x does not change (e.g., $0 \le i < l^{pos}$). This means that each value of x has $l^{(pos-1)}$ LGNs satisfying $x \equiv x' + base' - base \pmod{l}$. The number of x is $l^{(k/l-pos)}$, thus there are $l^{(k/l-1)}$ LGNs satisfying $g_{i,j} = g_{i,j'}$; Case 2) $pos < pos'$: Same as Case 1; Case 3) $pos = pos'$: not possible since $j' \not\equiv j \pmod{k/l}$. Since $0 \le j' < k$ and $j' \not\equiv j \pmod{k/l}$, the number of working data disks is $k - l$. That is, each of $k - l$ data disks stores $l^{(k/l-1)}$ data chunks for reconstructing $l^{(k/l)}$ data chunks of j-th data disk. Therefore, Property 2 is satisfied and Theorem 1 concludes. □

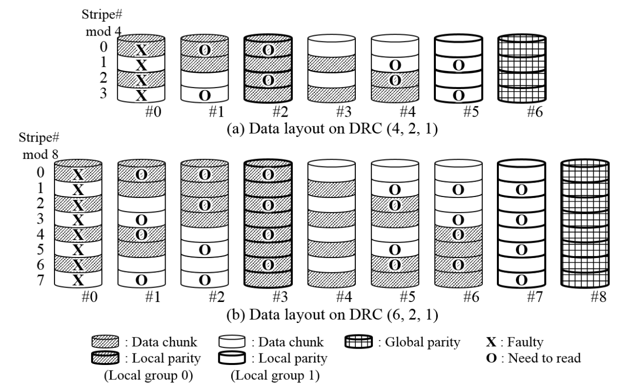

Fig. 6 Group shuffling on DRC (4, 2, 1) and DRC (6, 2, 1). In both cases, two local parity groups exist: local groups 0 and 1. Two local parity groups are defined in each stripe.

Figure 6(a) provides an example of group shuffling on DRC (4, 2, 1) in which the layout pattern is repeated every four stripes. DRC (6, 2, 1) is shown in Fig. 6(b) in which the layout pattern is repeated every eight stripes.

We apply local parity rotation every $l^{(k/l)}$ stripes. If local parity rotation is applied every stripe, Property 2 cannot be met. Figure 7(a) shows an example of data layouts in which local parity rotation is applied every four stripes, because $l^{(k/l)}$ becomes four in DRC (4, 2, 1). For stripes of 0, 1, 2, and 3, the data layouts are configured by Algorithm 1. For stripes of 4, 5, 6, and 7, local parity rotation is applied and the remaining data layouts are configured by Algorithm 1 as well.

In addition to local parity rotation, we consider global

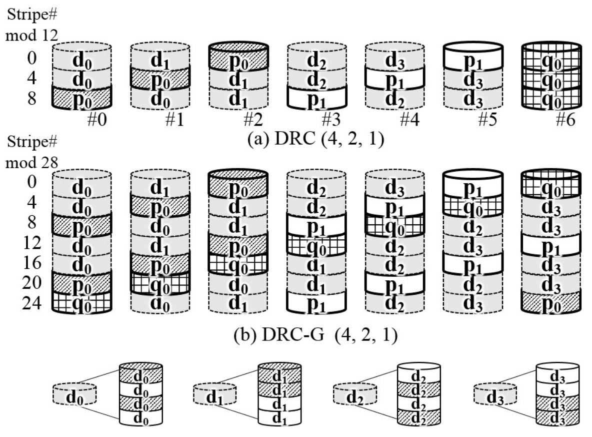

Fig. 7 Parity-rotated data layout on DRC (4, 2, 1) and DRC-G (4, 2, 1). In both cases, parity rotation is applied every four stripes.

parity rotation. Figure 7(b) shows an example of data layouts using global parity rotation in DRC (4, 2, 1), called DRC-G. In DRC-G,k+l–1 disks are involved in data reconstruction, whereas k disks are involved in DRC. Thus, we can easily predict that the performance of data reconstruction of DRC-G is better than that of DRC. However, with respect to reliability, DRC-G cannot recover any of three disk failures. This means that the corresponding p_d value of DRC-G is 0. For example, the p_d for DRC (4, 2, 1) is 0.43, whereas for DRC-G (4, 2, 1) it is 0.

3.2 Analysis of Performance and Reliability

3.2.1 Degraded Read Performance

We consider two cases: degraded reads with and without recovery processes running. The first case is the performance of full degraded reads (denoted as P_{FDR}). This represents the relative performance of degraded reads over normal reads without a recovery process. The second case is the performance of effective degraded reads (denoted as P_{EDR}). This represents the relative performance of degraded reads over normal reads while the recovery process is progressing.

P_{FDR} is calculated by analyzing the performance degradation in overloaded disks during disk failures. For a simple analysis, the DRC (4, 2, 1) configuration is assumed. Note that P_{FDR} may vary depending on the location of faulty disks. However, in this study, we focus on the worst performance because of its direct effect on customer SLAs. Therefore, the location of faulty disks resulting in the worst P_{FDR} is considered in each case.

Case 1. A single disk failure: assume a disk #0 is failure while each of the disks #0, #1, #3, and #4 receives four read requests in Fig. 8(a). Each of the overloaded disks #1 and #4 performs two read operations to reconstruct the four data chunks of faulty disk #0. This means that each of the overloaded disks performs six read operations while processing four read requests. Thus, P_{FDR} is $4/6 \sim 0.67$. In other words, the number of read operations additionally issued to overloaded disks for data reconstruction, denoted as AR, is $2/4$ per read request. Thus, P_{FDR} is $(1+2/4)^{-1} \sim 0.67$.

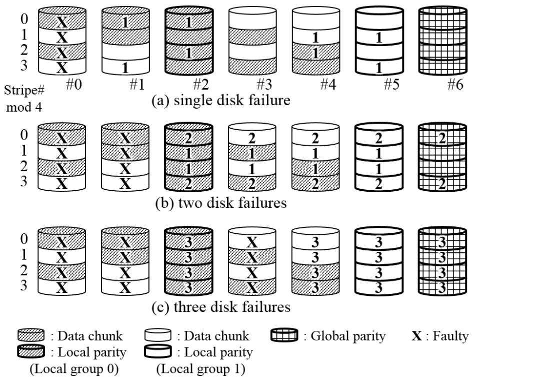

Fig. 8 The number of read operations additionally issued to disks in order to reconstruct the data chunks of faulty disks in DRC (4, 2, 1).

Table 1 The number of read operations additionally issued to overloaded disks for data reconstruction (AR) on DRC (k, l, r), where k is any number divisible by l, l is a number between 2 and 4, and r is any number. Note that l is the only attribute that affects AR.

		Number of Local Parities (l)		
		2	3	4
Number of Faulty Disks (F)	1	2/4	3/9	4/16
	2	6/4	10/9	14/16
	3	12/4	21/9	30/16
	4	X	36/9	52/16
	5	X	X	80/16

Case 2. Two disk failures: assume disks #0 and #1 are failure while progressing the same workload of Case 1 in Fig. 8(b). Each of the overloaded disks #3 and #4 performs six read operations to reconstruct the eight data chunks of faulty disks #0 and #1 while processing four read requests. Note that, if the data chunks of two faulty disks belong to the same local group (e.g., stripes of 0 and 3), all surviving disks are read in order to reconstruct a single data chunk of faulty disk. Thus, AR is 6/4 per read request, and P_{FDR} is $(1+6/4)^{-1} \sim 0.4$.

Case 3. Three disk failures: assume disks #0, #1 and #3 are failure while progressing the same workload of Case 1 in Fig. 8(c). The overloaded disk #4 performs twelve read operations to reconstruct the twelve data chunks of faulty disks #0, #1 and #3 while processing four read requests. Thus, AR is 12/4 per read request, and P_{FDR} is $(1+12/4)^{-1} \sim 0.25$.

In the same manner, we calculate ARs of DRC using three local parities and four local parities in Table 1. We now generalize the manner in which to calculate AR of DRC with an arbitrary number of faulty disks. Assume the number of faulty disks is F. In Table 1, ARs of DRC can be derived as $(F + 1) \times F/l^2$, $(2 \times F + 1) \times F/l^2$, and $(3 \times F + 1) \times F/l^2$ when ls are 2, 3, and 4, respectively. We can then calculate AR of DRC and P_{FDR} as follows:

$$AR_{DRC} = \frac{((l-1) \times F + 1) \times F}{l^2} \qquad (1)$$

$$P_{FDR} = \frac{1}{1 + AR} \qquad (2)$$

For a comparison, we also calculate AR of RAID-6 and

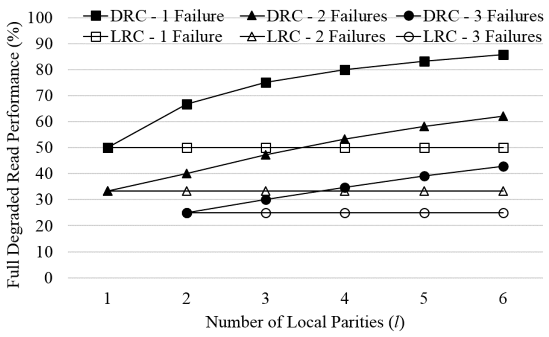

Fig. 9 Full degraded read performance (P_{FDR}) of DRC (k, l, r) and LRC (k, l, r), where k is any number divisible by l, l is a number between 1 and 6, and r is any number. Note that l is the only attribute that affects P_{FDR}. (P_{FDR} of RAID-6 (k, r) is the same as that of LRC (k, 1, r).)

LRC. If a single data disk fails, AR of RAID-6 and LRC is 1. Thus, P_{FDR} is $(1 + 1)^{-1} = 0.5$. Moreover, AR of LRC does not change in a configuration having a varying number of local parity groups. Therefore, AR of RAID-6 and LRC equation is obtained as follows:

$$AR_{RAID\text{-}6,LRC} = F \tag{3}$$

We can then calculate P_{FDR} by Eq. (2) once we determine AR.

Figure 9 shows how P_{FDR} of DRC and LRC vary according to the number of local parities that include multiple failures. P_{FDR} of RAID-6 is exactly the same as that of LRC, using one local parity. We observe that, unlike in LRC with the same conditions, P_{FDR} of DRC improves considerably by increasing the number of local parities.

Let us now consider recovery processes in order to calculate P_{EDR}. The bandwidth allocated for recovery processes is denoted as B_{rp}. The more bandwidth allocated to recovery processes, the less is available to handle outstanding read requests in surviving disks. The recovery processes consume $B_{rp} \times AR$ of read bandwidth from each surviving disk and consume B_{rp} of write bandwidth from a replacement disk. Therefore, B_{rp} can be increased to $\min(B_{read}/AR, B_{write})$, where B_{read} is the maximum read bandwidth of a disk and B_{write} is the maximum write bandwidth of a disk. We denote U_{rp} as the utilization of overloaded disks caused additionally by the recovery process. Then, U_{rp} is derived as Eq. (4). Note that DRC requires less U_{rp} than does LRC because DRC generates smaller AR for reconstruction in recovery process than does LRC. P_{EDR} is derived as Eq. (5) because the recovery process interferes with degraded reads. Note that P_{EDR} represents performance of degraded reads while the recovery process is progressing.

$$U_{rp} = \frac{B_{rp} \times AR}{B_{read}} \quad (0 < B_{rp} < \min(\frac{B_{read}}{AR}, B_{write})) \tag{4}$$

$$P_{EDR} = P_{FDR} * (1 - U_{rp}) \tag{5}$$

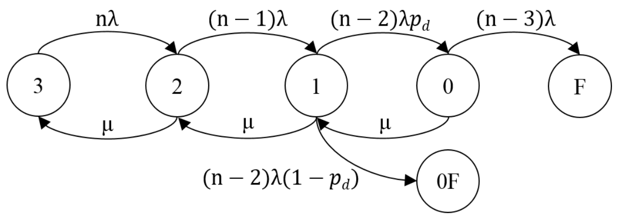

Fig. 10 Markov model for reliability analysis of DRC (k, 2, 1); n denote the total number of disks, λ the failure rate of a single disk, and μ the repair rate of a disk.

3.2.2 Reliability Analysis

Mean time to data loss (MTTDL) is used to calculate the reliability of storage systems [10], [15]. To analyze MTTDL, a standard Markov model is generally used, assuming that disk failures are independent. For example, Fig. 10 plots the Markov model for reliability analysis of DRC (k, 2, 1). The number of each state represents the number of parity chunks available to that state.

Let n, λ, and μ denote the total number of disks (i.e., $k+2+1$), the failure rate of a single disk, and the repair rate of a disk, respectively. Two transitions from *State 1* exist in which two disk failures have already occurred. Depending on the recoverability ratio p_d, an additional disk failure can be recovered (i.e., transitioned to *State 0*) or cannot be recovered (i.e., transitioned to *State 0F*).

To measure the failure rate of an array storage system (λ_{array}), we calculate the failure rate of the transition from *State 3* to *State F* ($\lambda_{3\to F}$) as well as the transition from *State 3* to *State 0F* ($\lambda_{3\to 0F}$). In addition, we compute the sum of the two rates and thus, obtain Eq. (6) and Eq. (7), respectively respectively (please refer [15] that explains how to derive the equations). Combining these two equations by considering p_d allows us to obtain Eq. (8).

$$\lambda_{3\to F} = \frac{n(n-1)(n-2)(n-3)\lambda^4 p_d}{\mu^3} \tag{6}$$

$$\lambda_{3\to 0F} = \frac{n(n-1)(n-2)\lambda^3(1-p_d)}{\mu^2} \tag{7}$$

$$\lambda_{array} = \frac{n(n-1)(n-2)\big(\lambda^4(n-3)p_d + \lambda^3\mu(1-p_d)\big)}{\mu^3} \tag{8}$$

We can then derive MTTDL as $\lambda_{array}{}^{-1}$. Note that the mean time to repair (MTTR) is μ^{-1}, and the mean time to failure (MTTF) is λ^{-1}. We can calculate MTTDL as follows:

$$MTTDL = \frac{MTTF^4}{n(n-1)(n-2)\big(MTTR^3(n-3)p_d + MTTR^2 MTTF(1-p_d)\big)} \tag{9}$$

Typically, MTTF is provided by disk manufacturers. MTTR is determined by disk size (denoted as S_{disk}) and

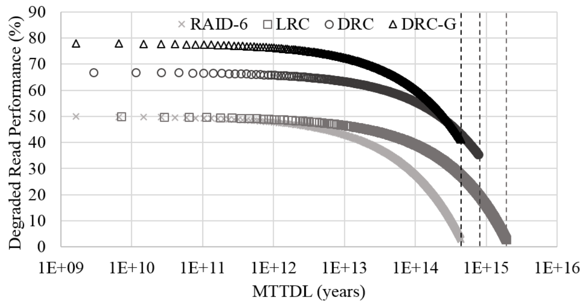

Fig. 11 Relationship between degraded read performance and MTTDL on RAID-6 (4, 2), LRC (4, 2, 1), DRC (4, 2, 1), and DRC-G (4, 2, 1). Degraded read performance is normalized to the normal read performance of RAID-6.

the amount of bandwidth allocated to the recovery process. Thus, MTTR is calculated as follows:

$$MTTR = \frac{S_{disk}}{B_{rp}} \tag{10}$$

3.2.3 Trade-Off between Performance and Reliability

Because the bandwidth is a limited resource, trade-off exists between degraded read performance and the reliability of an array storage system. In this section, we analyze the degree to which DRC effectively controls the trade-off between degraded read performance and reliability.

We assume DRC (4, 2, 1) and DRC-G (4, 2, 1). LRC (4, 2, 1) and RAID-6 (4, 2) configurations are considered for comparison. Note that the p_d of DRC, DRC-G, LRC, and RAID-6 are 0.43, 0, 0.77, and 0, respectively. Based on the specification of SSD [16], we set B_{read} to 550 MB/s, B_{write} to 520 MB/s, S_{disk} to 240 GB, and MTTF to 2 million hours. We assume a single disk failure.

Figure 11 shows variations in degraded read performance based on MTTDL. Degraded read performance is calculated by means of P_{EDR}, which is based on the amount of bandwidth of the recovery process necessary to satisfy the given MTTDL. In this graph, degraded read performance is normalized to the performance of normal reads of RAID-6. DRC and DRC-G achieve much better degraded read performance on the same MTTDL than does LRC. However, because the bandwidth of the recovery process is limited to the maximum read/write bandwidth of the disk, DRC and DRC-G can support MTTDL by as much as 7.75E+14 and 4.42E+14, respectively. This means that LRC can only support MTTDL from 7.75E+14 to 1.92E+15. However, we argue that this range of MTTDL is not applicable to primary array storage systems because degraded read performance drops to less than 20% against normal read performance.

4. Evaluation

We implemented the proposed erasure codes, DRC and DRC-G, in a Linux MD layer [17]. For comparison purposes, LRC was also implemented. By default, Linux MD supports RAID-6. The Linux kernel version is 3.18.9. Test configurations include RAID-6 (4, 2), LRC (4, 2, 1), DRC (4, 2, 1), and DRC-G (4, 2, 1). RAID parameters such as chunk size and queue depth are set to 128 KB and 128 requests, respectively. For data reconstruction and the write operation in the Linux MD layer, we must determine the amount of memory (e.g., the number of page entries) and the number of MD (kernel) threads. We reserve 8192 page entries per disk and set 16 MD threads.

We tested every software operation on a DELL 720xd server with two CPUs of Intel Xeon E5-2630 and 96 GB memory of DDR3. The server was equipped with a host bus adapter of LSI SAS 9300-8i [18] that supports bandwidth of as much as 8 GB/s by eight lanes of PCIe 3.0 connectivity. A Samsung SSD 850 Pro (256GB) [16] was used as a disk.

4.1 Synthetic Workloads

We used fio-2.2.10 (flexible I/O tester) [19] as a benchmark with different access patterns. Each result was averaged based on 10 experiments conducted under the same benchmark. Prior to each experiment, secure erase operations were executed for every SSD to avoid any interference.

First, we measured the performance of degraded reads and writes without recovery processes running. To utilize full bandwidth, we tested throughput using eight jobs on the workload of random 4 KB reads and one job on the other three workloads: sequential 128 KB reads, random 4 KB writes, and sequential 128 KB writes. We removed disks determined to be faulty based on the number of disk failures, and we performed each workload. The results of each test performance varied depending on the location of faulty disks; therefore, we specified the performance result using a range. We assumed that, on LRC and DRC, a disk was exclusively used to store global parity.

The degraded read performance on random reads of 4 KB is shown in Fig. 12(a). In the case of a single disk failure, RAID-6 and LRC showed nearly 50% of normal read performance. By contrast, DRC and DRC-G showed 68%. In the case of two disk failures, the average performances of LRC and DRC remained nearly the same, but we observed that DRC was less sensitive to faulty disk positions than was LRC. Moreover, the worst case performance of DRC was 20% higher than that of LRC.

Figure 12(b) shows the degraded read performance on sequential reads of 128 KB. In the case of a single disk failure, it shows nearly the same results as those of a random read. Sequential reads workload incurs read requests from data chunks on the same stripe. Requests merging may be considered between direct read requests and read requests for reconstruction. However, we observed few possible merges because of different sizes. For example, a direct request reads a length of data of 128 KB and a request for reconstruction reads a length of data of 4 KB. In the case of two disk failures, we observed requests merging between read requests for reconstruction. Thus, degraded read performance was increased more than did random read. The worst performance of DRC remained 10% higher than that of LRC.

The degraded write performance on random writes of 4 KB is shown in Fig. 13(a). In general, write requests require parity updates, that is, both local and global parity. Thus, as expected, the advantage of fast reconstruction is not considerable in DRC. However, DRC-G performs better than DRC because of its global parity rotation. Because of space limitations, we omit a detailed behavior analysis.

Figure 13(b) shows the degraded write performance on sequential writes of 128 KB. In the case of sequential writes of large data, reconstruction may not be required to calculate a new parity. Therefore, as shown in Fig. 13(b), the results of all codes reveal nearly the same performance.

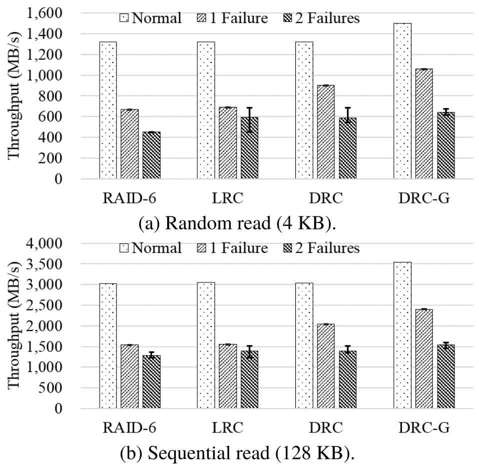

Fig. 12 Degraded read performance without a recovery process running on RAID-6 (4, 2), LRC (4, 2, 1), DRC (4, 2, 1), and DRC-G (4, 2, 1). (The results of three disk failures are omitted because each code handles three-disk-failure cases differently; for example, RAID-6 and DRC-G cannot recover any three-failure case, whereas LRC and DRC can recover 77% and 43% of three-failure cases, respectively.)

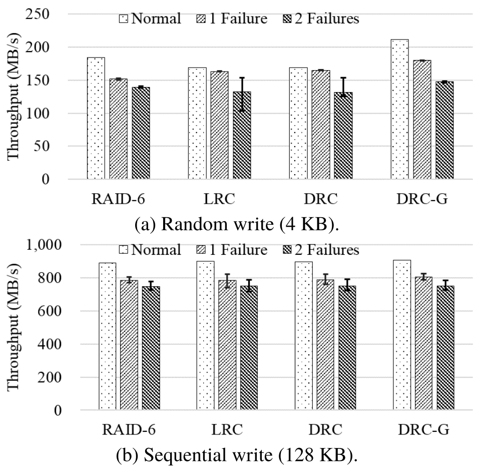

Fig. 13 Degraded write performance without a recovery process running on RAID-6 (4, 2), LRC (4, 2, 1), DRC (4, 2, 1), and DRC-G (4, 2, 1). (The results of three disk failures are omitted because each code handles three-disk-failure cases differently; for example, RAID-6 and DRC-G cannot recover any three-failure case, whereas LRC and DRC can recover 77% and 43% of three-failure cases, respectively.)

Second, we measured the performance of degraded reads while maintaining the same reliability of an array storage system. To maintain the same reliability, different amounts of bandwidth are allocated to the recovery process in each system of RAID-6, LRC, and DRC. To satisfy the 2.5E+13 MTTDL, the recovery process of RAID-6, LRC, DRC, and DRC-G are configured to have 125, 60, 94, and 125 MB/s, respectively. We removed one disk as faulty, and we replaced it with a new disk to start the recovery process. Simultaneously, we performed a random read (4 KB) workload. The results are shown in Fig. 14.

Each performance improved gradually as time passed, because already recovered data chunks (as a result of the recovery process) do not need to be reconstructed. RAID-6 starts with 500 MB/s and was 38% of the normal read performance of 1310 MB/s. The lower bound of performance, which was normalized to the normal read performance of RAID-6, was 59%, 65%, and 47% on DRC, DRC-G, and LRC, respectively. These results are accurate under a 4% error against results of degraded read performance, which is calculated in Sect. 3.2.3. In addition, the recovery process of DRC and DRC-G was complete earlier than that of LRC. The result showed that DRC returns to normal mode 36% earlier than LRC, and DRC-G was 52% earlier than LRC.

4.2 Real-World Workloads

Four real-world workload traces (downloaded from the Storage Performance Council [20]) were used to evaluate the effectiveness of DRC. The characteristics of each trace are listed in Table 2. The experiments were conducted while maintaining the same reliability of an array storage system.

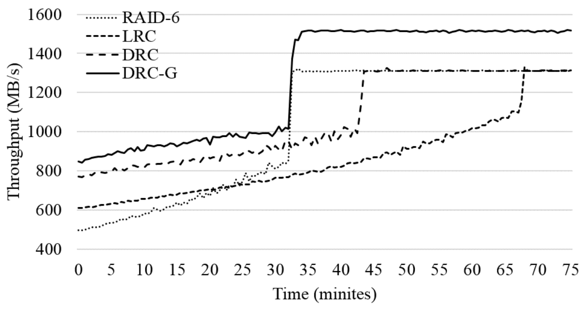

Fig. 14 Performance of a random read (4 KB) workload running on RAID-6 (4, 2), LRC (4, 2, 1), DRC (4, 2, 1), and DRC-G (4, 2, 1) while a recovery process progresses during a single disk failure. To maintain the same reliability for comparative purposes, the bandwidth allocated to recovery process in RAID-6, LRC, DRC, and DRC-G is 125, 60, 94, and 125 MB/s, respectively.

Table 2 Trace characteristics.

Trace name	Read ratio (%)	Average request size (KB)	Amount of requests (GB)	Working set size (GB)
WebSearch1	99	15.07	65.82	994
WebSearch2	99	15.4	62.59	1508
Financial1	15	3.37	17.19	182
Financial2	78	2.39	8.43	66

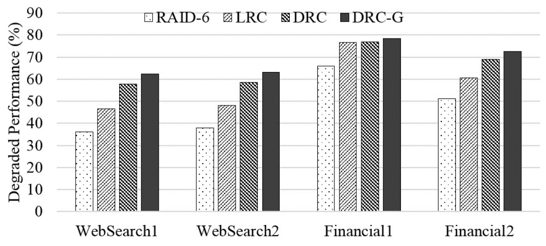

Fig. 15 Degraded performance of real-world workloads running on RAID-6 (4, 2), LRC (4, 2, 1), DRC (4, 2, 1), and DRC-G (4, 2, 1) while a recovery process progresses during a single disk failure. Degraded performance is normalized to the normal performance of RAID-6. To maintain the same reliability for comparative purposes, the bandwidth allocated to recovery process in RAID-6, LRC, DRC, and DRC-G is 125, 60, 94, and 125 MB/s, respectively.

Table 3 Disk utilization (%) of a sequential read (128 KB) workload running on RAID-6 (4, 2), LRC (4, 2, 1), DRC (4, 2, 1), and DRC-G (4, 2, 1) during disk failures.

(a) a single disk failure (#0)

	#0	#1	#2	#3	#4	#5	#6
RAID-6	F	99.5	95.84	96.28	95.95	95.71	X
LRC	F	99.36	99.26	55.31	55.59	56.02	0.03
DRC	F	99.35	99.29	69.73	99.39	99.6	0.03
DRC-G	F	99.66	96.01	78.23	89.08	95.88	95.52

(b) two disk failures (#0 and #1)

	#0	#1	#2	#3	#4	#5	#6
RAID-6	F	F	99.41	99.52	99.31	99.82	X
LRC	F	F	99.29	99.55	99.28	99.25	77.75
DRC	F	F	99.23	99.32	99.22	99.29	61.91
DRC-G	F	F	99.11	99.49	99.15	99.24	99.46

Figure 15 shows the degraded performance of real-world workloads during a single disk failure. In this graph, results are normalized to normal performance (i.e., under no disk failure) of RAID-6. WebSearch1 and WebSearch2 are read-dominant over 99%, thus DRC and DRC-G improved considerably with respect to degraded performance. In WebSearch1, the degraded performance of RAID-6, LRC, DRC, and DRC-G showed 36.1%, 46.4%, 57.9%, and 62.4%, respectively. This means that the degraded performance of DRC improved by 60% over that of RAID-6 and 25% over that of LRC. In addition, DRC-G achieved 72% improved performance over that of RAID-6 and 35% over that of LRC. A similar improvement appears in the case of Financial2, which is also read-intensive but issues small-sized requests. By contrast, because Financial1 is a write-intensive workload, the performance advantage of DRC over LRC was insignificant: less than 3%.

Although all experiments were conducted using a solid state drive (SSD), the results do not depend on SSD because we observed identical situations for HDD and SSD.

4.3 Disk Utilization

We measured the disk utilization of degraded reads during disk failures. The testing environment was same as the experiment of sequential 128 KB reads in Fig. 12(b) of Sect. 4.1.

Table 3(a) shows the disk utilization on sequential reads of 128 KB during a failure of disk #0. The degraded read performance of RAID-6, LRC, DRC, and DRC-G was 1545, 1555, 2039, and 2402 MB/s, respectively. Note that, for a single data reconstruction, four disks are read on RAID-6 and two disks are read on LRC, DRC, and DRC-G. On LRC, only disks #1 and #2 showed full utilization while disks #3, #4, and #5 showed nearly 55% utilization. Unlike on LRC, disks #1, #2, #4, and #5 showed full utilization on DRC. This means that four disks are equally involved in data reconstruction during reconstructing multiple data. On DRC-G, the disk #6 is additionally utilized.

Table 3(b) shows the disk utilization on sequential reads of 128 KB during failures of disks #0 and #1. The degraded read performance of RAID-6, LRC, DRC, and DRC-G was 1229, 1230, 1349, and 1456 MB/s, respectively. Note that the results were the worst case performance of each erasure code. In the worst case, five disks are read for every data reconstruction on LRC. However, on DRC, only two disks are read in half of all data reconstruction cases. Thus, the performance of DRC showed 10% higher than that of LRC, although the disk #6 of DRC showed lower utilization than that of LRC. All disks showed full utilization on DRC-G.

5. Related Works

Erasure codes for fast reconstruction: Two approaches are used to support fast reconstruction. The first approach [2]–[6] involves reducing the amount of data read from disks for reconstruction. This approach focuses on finding coding coefficients for reconstruction or optimizing a reconstruction algorithm while maintaining the *Maximum Distance Separable* (MDS) property. Regenerating Codes (RGC) [2] find coding coefficients. RGCs remain an active and open research topic. RGC-based F-MSR [3] achieves cost-effective reconstruction for a single failure, and Butterfly codes [4] have recently been applied to and verified on two popular distributed storage systems (HDFS and Ceph). RDOR [5] is the recovery optimized codes of row-diagonal parity (RDP). Rotated RS [6] modifies decoding algorithms for fast reconstruction. However, existing practical solutions [3], [6] achieve only approximately 20-30% savings in terms of I/O and bandwidth, considerably less than DRC (more than 50%) that is non-MDS.

The second approach, using non-MDS codes [7]–[11], involves reducing the number of disks for reconstruction, as in DRC. WEAVER codes [7] are extremely efficient, but unfortunately require high storage overhead (2x and greater). The storage cost of both HoVer codes [8] and Stepped Combination codes [9] is less than 2x. Among these codes, LRC [10], [11] is known to offer the best (or optimal) trade-off between storage overhead, fault tolerance, and the number of disks involved in reconstruction. LRC is adopted and verified on two distributed storage systems (Windows Azure Storage [10] and HDFS [12]). However, LRC has a problem of uneven disk utilization, causing low

reconstruction performance on array storage systems.
Reconstruction optimization: PRO [21] recovers popular data to diminish performance degradation during disk failures by means of workload analysis. D-GRAID [22] recovers only blocks that contain live data. These two methods can improve degraded read performance based on workloads because the recovery process interferes with degraded reads. We believe that they can also be applied to our codes. Partial Parallel Repair (PPR) [23] divides reconstruction operations into smaller partial operations and then schedules them on multiple nodes in order to mitigate congested network traffic for data reconstruction. However, the benefits of PPR only become obvious when network traffic causes a bottleneck.

6. Conclusion

In this paper, we proposed erasure codes called DRC that support fast reconstruction for array storage systems. The main idea behind DRC is group shuffling, which reduces the performance degradation of overloaded disks as many as possible in a degraded mode. We showed that, in array storage systems, DRC achieves a better trade-off between degraded read performance and reliability than does LRC. Through experiments using real-world workloads, we showed that the degraded performance of DRC-G improved by 72% over that of RAID-6 and 35% over that of LRC under the same reliability. In addition, we showed that DRC-G reduces the recovery process completion time by as much as 52% compared to that of LRC.

References

[1] D.A. Patterson, G. Gibson, and R.H. Katz, "A case for redundant arrays of inexpensive disks (RAID)," Proc. 1988 ACM SIGMOD International Conference on Management of Data, pp.109–116, 1988.
[2] A.G. Dimakis, P. Godfrey, Y. Wu, M.J. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," IEEE Trans. Information Theory, vol.56, no.9, pp.4539–4551, 2010.
[3] Y. Hu, H.C. Chen, P.P. Lee, and Y. Tang, "NCCloud: applying network coding for the storage repair in a cloud-of-clouds," USENIX FAST, 2012.
[4] L. Pamies-Juarez, F. Blagojevic, R. Mateescu, C. Guyot, E.E. Gad, and Z. Bandic, "Opening the chrysalis: on the real repair performance of MSR codes," USENIX FAST, 2016.
[5] L. Xiang, Y. Xu, J. Lui, and Q. Chang, "Optimal recovery of single disk failure in RDP code storage systems," ACM SIGMETRICS Performance Evaluation Review, pp.119–130, 2010.
[6] O. Khan, R. Burns, J. Plank, and W. Pierce, "Rethinking erasure codes for cloud file systems: minimizing I/O for recovery and degraded reads," USENIX FAST, 2012.
[7] J.L. Hafner, "WEAVER codes: highly fault tolerant erasure codes for storage systems," USENIX FAST, 2005.
[8] J.L. Hafner, "HoVer erasure codes for disk arrays," IEEE Proc. of DSN, pp.217–226, 2006.
[9] K.M. Greenan, X. Li, and J.J. Wylie, "Flat XOR-based erasure codes in storage systems: constructions, efficient recovery, and tradeoffs," IEEE Mass Storage Systems and Technologies, 2010.
[10] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, et al., "Erasure coding in Windows Azure storage," USENIX Annual Technical Conference, 2012.
[11] D.S. Papailiopoulos and A.G. Dimakis, "Locally repairable codes," IEEE Trans. Information Theory, vol.60, no.10, pp.5843–5855, 2014.
[12] M. Xia, M. Saxena, M. Blaum, and D.A. Pease, "A tale of two erasure codes in HDFS," USENIX FAST, 2015.
[13] E. Miller, "Inside the Pure Storage flash array: building a high performance, data reducing storage system from commodity SSDs," Presentation on IEEE Mass Storage Systems and Technologies, 2014.
[14] Kaminario, "K2 V5 Architecture," White paper, Aug. 2015.
[15] Q. Xin, E.L. Miller, T. Schwarz, D.D. Long, S.A. Brandt, and W. Litwin, "Reliability mechanisms for very large storage systems," IEEE Mass Storage Systems and Technologies, 2003.
[16] Samsung SSD 850 PRO data sheet, http://www.samsung.com/semiconductor/minisite/ssd/downloads/document/Samsung_SSD_850_PRO_Data_Sheet_rev_2_0.pdf.
[17] Linux RAID, https://raid.wiki.kernel.org/index.php/Linux_Raid.
[18] Avago SAS 9300 12Gb/s SAS host bus adapter family product brief, https://docs.broadcom.com/docs/12352000.
[19] Flexible I/O tester, https://github.com/axboe/fio.
[20] UMassTraceRepository. Search engine I/O and OLTP application I/O, http://traces.cs.umass.edu/index.php/Storage/Storage.
[21] L. Tian, D. Feng, H. Jiang, K. Zhou, L. Zeng, J. Chen, et al., "PRO: a popularity-based multi-threaded reconstruction optimization for RAID-structured storage systems," USENIX FAST, 2007.
[22] M. Sivathanu, V. Prabhakaran, A.C. Arpaci-Dusseau, and R.H. Arpaci-Dusseau, "Improving storage system availability with D-GRAID," ACM Trans. Storage, vol.1, no.2, pp.133–170, 2005.
[23] S. Mitra, R. Panta, M.-R. Ra, and S. Bagchi, "Partial-parallel-repair (PPR): a distributed technique for repairing erasure coded storage," ACM European Conference on Computer Systems, 2016.

Baegjae Sung received a B.S. degree in Computer Science and Engineering from Inha University, Korea in 2006, and a M.S. degree in Information Technology from POSTECH, Korea in 2009. He is currently a Ph.D. candidate in the Department of Computer Science and Engineering, POSTECH, Korea. His research interests include storage systems, virtualization technology, and embedded systems.

Chanik Park received a B.S. degree in 1983 from Seoul National University, and a M.S. degree and Ph.D. in 1985 and 1988, respectively, from Korea Advanced Institute of Science and Technology. Since 1989, he has been working for POSTECH, where he is currently a professor in the Department of Computer Science and Engineering. He was a visiting scholar with the Parallel Systems group in the IBM Thomas J. Watson Research Center in 1991, and a visiting professor with the Storage Systems group in the IBM Almaden Research Center in 1999. He has also visited Northwestern University and Yale University in 2008 and 2015, respectively. He has served at a number of international conferences as a program committee member. His research interests include storage systems, operating systems, and system security.