The Journal of Supercomputing (2019) 75:4638–4656
https://doi.org/10.1007/s11227-018-2663-4

Cost analysis of erasure coding for exa-scale storage

Dong-Oh Kim[1] · Hong-Yeon Kim[1] · Young-Kyun Kim[1] · Jeong-Joon Kim[2]

Published online: 31 October 2018

Abstract
With the increasing demand for mass storage, research on exa-scale storage is actively underway. When the scale of storage grows to the exa-scale, the space efficiency becomes very important. To maintain the storage reliability and improve the space efficiency, we have begun to introduce erasure coding instead of replication. However, erasure coding has many I/O performance degradation factors such as Parity Calculation, degraded I/O, Data Distribution cost, etc., whereas the existing research mainly focuses on improving the performance of the Parity Calculation.In this study, we identified the issues and bottlenecks of using erasure coding in real storage. First, we measured the I/O performance of various erasure codes to find the suitable erasure codes for real storage. Next, we analyzed the execution time for each processing step when I/O was performed and the issues when erasure coding was used in storage. Finally, we predicted the cost of EC-based I/O processing in the exa-scale storage and identified the expected problems.

Keywords Erasure coding · Parity calculation cost · Exa-scale storage · ExSTorus-FS · Degradation factor

✉ Jeong-Joon Kim
jjkim@kpu.ac.kr

Dong-Oh Kim
dokim@etri.re.kr

Hong-Yeon Kim
kimhy@etri.re.kr

Young-Kyun Kim
kimyoung@etri.re.kr

[1] High Performance Computing Research Department, Electronics and Telecommunications Research Institute, Daejeon, Korea

[2] Department of Computer Engineering, Korea Polytechnic University, Siheung, Gyeonggi-do, Korea

1 Introduction

Recently, with the increasing number of fields to generate and analyze a large amount of data, the demand for large-capacity storage has increased. In particular, research on exa-scale storage is actively underway. Exa-scale storage has increased throughput, capacity, system size, power usage, number of files, frequency of failure, etc., in the range of tens to hundreds of times compared to the existing storage. Therefore, various studies are required to efficiently handle the rapidly increasing of these demands [9, 15, 20, 22].

In particular, replication has been widely used to ensure reliability in storage. Because replication copies data to many different devices, it requires more physical storage than is actually required. For example, triple replication requires three times more physical storage than needed. When replication is used on exa-scale storage, the scale of the system will grow to several times the required capacity, which will significantly increase the cost of deployment and management. Therefore, space efficiency is becoming notably important to reduce the size of the system in exa-scale storage [4, 14].

Erasure coding (EC) is a method of encoding data using an erasure code and recovering the original data by decoding upon data loss [25]. EC significantly improves the space efficiency. For example, 8 + 2 EC guarantees identical failures recovery as a triple redundancy but more than doubles the space efficiency. EC is divided into the client-based encoding and data-server-based encoding depending on the position where the parity is calculated. GlusterFS [7] is a client-based encoding file system; HDFS [1] and Ceph [2] are data-server-based encoding file systems. In this study, the data-server-based encoding was selected to minimize the network communication cost between the client and the data server with considerations for thin clients.

However, although EC is highly space efficient, there are I/O performance degradation factors such as the Parity Calculation, Data Distribution cost, small I/O problem, read-modify-write, and degraded I/O (repair I/O) [8, 21, 31, 33]. Because of these performance issues, the storage industry is recommending EC as storage for Cold I/O, such as backup and archive [5, 8, 30]. In the recent years, various technologies such as single instruction multiple data (SIMD) and multi-channel I/O have been introduced to solve this problem.

The existing research has mainly focused on improving the performance of the Parity Calculation in erasure coding. In [3, 10, 23], the authors proposed parallel erasure coding to reduce the overall erasure coding processing time. It also uses multi-core or GPU to efficiently handle parallel processing. In [16], the authors proposed recovery using a minimal amount of data. In [32], the authors proposed the Shingled Erasure Code (SHEC), which was designed to efficiently recover from multiple disk failures and could be customized and user-adjustable with various local parity group layouts.

Previous studies have not mentioned the Data Distribution cost, small I/O problems, etc., which may be a problem in the actual storage. In this study, we identified the problems and bottlenecks when erasure coding was used in real storage. First, we measured the write performance of various erasure codes to find the suitable erasure codes for real scale-out storage. Then, we checked the weight of the Parity Calculation

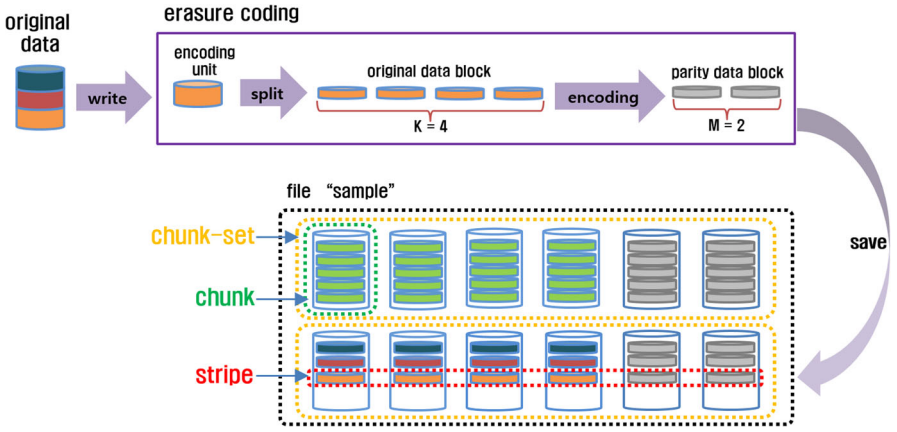

Fig. 1 Example of 4+2 EC volume

in EC and analyzed the write cost in the real storage to find the most important degradation factor.

Thus, we measured the write time for each processing step in the EC volume at various I/O sizes and networks speeds. Then, we analyzed them to identify the key problems when EC was applied to real storage and predicted the write time of the EC volume in the exa-scale storage and identified the expected problems.

2 Load analysis of EC in real storage

In this section, we discuss the measurement of the write performance of various erasure codes on real scale-out storage. We found the suitable erasure codes for real scale-out storage and checked the weight of the Parity Calculation in EC.

2.1 Concept of EC

EC is a method of encoding data using an erasure code and recovering the original data by decoding upon data loss. The encoding means Parity Calculation from the data. The encoding in the EC is performed on an encoding unit basis. EC divides the encoding unit into K original data blocks and generates M parity data blocks by encoding from K original data blocks. It is called $K+M$ EC. Figure 1 shows an example of the 4+2 EC volume to illustrate the configuration of the $K+M$ EC volume.

As shown in Fig. 1, in the 4+2 EC volume, the original data are stored in three stripes through three encoding operations. The encoding unit is divided into 4 original data blocks, and 2 parity data blocks are generated by the Parity Calculation. When data smaller than the encoding unit are entered, encoding is performed by filling the zero bit in the size of the encoding unit. A stripe is the minimum storage unit of the EC volume, which is a set of original data blocks and parity data blocks associated with a single encoding operation. A chunk represents a file of a specified size stored on a

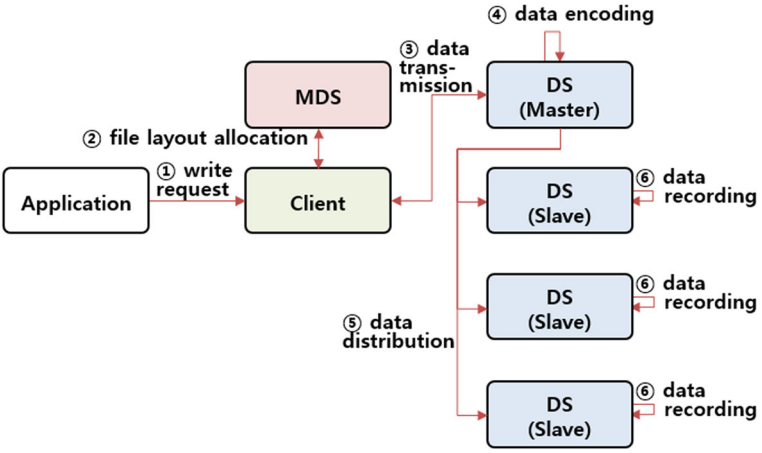

Fig. 2 Process of writing data in the EC volume of the ExSTorus-FS

storage device such as a disk and stores data blocks of the same location in a stripe. A chunk set is a set of chunks where a single stripe is divided and stored, and a file consists of one or more chunk sets. In Fig. 1, a file "sample" consists of 2 chunk sets. A chunk set consists of 4 original data chunks and 2 parity data chunks, and a stripe consists of 4 original data blocks and 2 parity data blocks.

2.2 ExSTorus-FS

ExSTorus-FS [18] is a scale-out distributed file system based on FUSE [6] for exa-scale storage, which supports Torus-based network configurations, EC-based Hot I/O, and multiple-MDS support [17]. ExSTorus-FS was developed on the basis of MAHA-FS, which is used for providing cloud storage services of hundreds of Peta-bytes [19].

ExSTorus-FS supports K [2, 4, 8, 16] $+M$ [1, 2, 4] EC volumes [17] and Reed Solomon (RS) [24], Cauchy Read Solomon (CRS) [28], Liberation (LIB) [27], Advanced Vector Extensions (AVX):CRS, and AVX2:CRS using the Jerasure library 1.2 [26], and Intel ISA library 2.10 [11, 12] by erasure codes. CRS and LIB are performance-optimized encoding methods that optimize RS. AVX:CRS and AVX2:CRS are an SIMD [29] instruction set based on CRS.

ExSTorus-FS is a data-server-based encoding file system. In this study, the data-server-based encoding was selected to minimize the network communication cost between the client and the data server with consider the thin clients.

ExSTorus-FS consists of the metadata server (MDS), data server (DS), and client. The MDS manages the metadata of files and monitors and manages the system. The DS processes the file I/O requests and periodically reports the status of the storage device and load of the server to the MDS. The client processes the user's request on the basis of POSIX by connecting it with ExSTorus-FS. Figure 2 shows the process of writing data in the EC volume of the ExSTorus-FS.

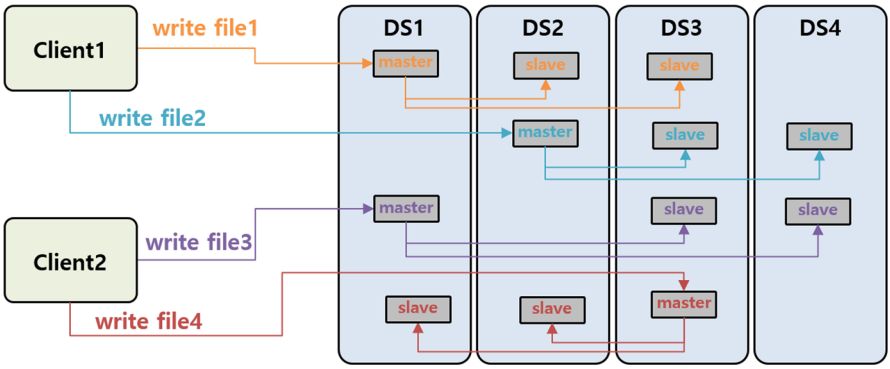

Fig. 3 Process of recording from several clients

As shown in Fig. 2, when the write is requested in the application, the client performs a file layout allocation through the MDS. The MDS checks the EC volume configuration to allocate the required chunks for the requested file. The client analyzes the file layout and transmits the written data to the DS to play the Master role. This DS is referred to as the Master. The Master is assigned one of the DSs with a chunk-by-chunk set and is responsible for the Parity Calculation and Data Distribution. After the data are encoded by the Master, the data and the parity are distributed to the DS to play the slave role. This DS is referred to as a Slave. A Slave is responsible for data recording to the storage. The Slave reports the result of the data recording to the Master. The Master collects the processing results of all Slaves and reports the final result to the client. Figure 3 shows the recording from several clients.

As shown in Fig. 3, a write request occurs to four files: file1 and file2 are executed in client1, and file3 and file4 are executed in client2. Data encoding sends a write request to the Master determined in the layout for each file. Master operates file1 and file3 in DS1, file2 in DS2, and file4 in DS3.

2.3 Write performance for various erasure codes

In this section, we describe the sequential write performance according to the erasure code. Experiments were performed on seven servers, each of which was equipped with two Intel Xeon CPUs E5-2609 2.40 GHz, 64 G memory, 1 G/10 G Ethernet, and ten 4 TB 7200-rpm HDDs, and the installed software included CentOS 7.0, ExSTorus-FS 1.0, and iozone 3.465. Experiments were performed on a 4 + 2 EC volume with a stripe size of 6 K and a chunk size of 64 M.

This experiment measured the write throughput and CPU usage while changing the erasure code to RS, CRS, LIB, AVX2:CRS, and NONE in the 4 + 2 EC volume of ExSTorus-FS. NONE indicated that no encoding function call was performed during write processing. These experiments were performed with 1 MDS, 6 DSs, and 1 client in 10 G Ethernet. For the performance comparison, we fixed the Master DS and performed writes in a single-threaded and a multi-threaded.

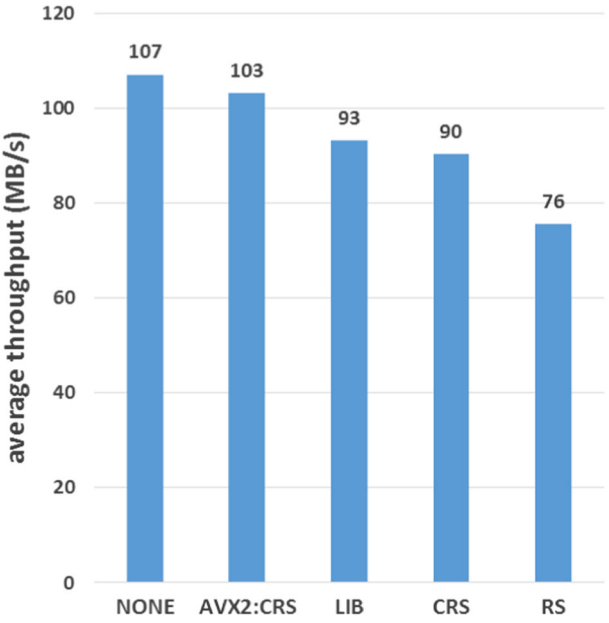

Fig. 4 Single-thread average throughput by erasure code

As shown in Fig. 4, AVX2:CRS had the most similar performance to NONE. The performance of LIB and CRS was reduced by 15%, and RS was reduced by 29% compared to NONE. In other words, the Parity Calculation accounted for 4–29% of the total IO. As shown in Fig. 5, AVX2:CRS, LIB, and CRS had similar utilization to NONE. RS differed from NONE by only 9%. The experimental results showed that HW encoding was better than SW encoding.

The multi-thread sequential write performance test was performed 10 times through the iozone [13], e.g., iozone -i 0 -r 128 K -s 5 G -t 10 -+n -w. To increase the parallel processing performance of ExSTorus-FS, we set the number of channels to 10 in the client. Figures 6 and 7 show the average throughput and average CPU usage according to the erasure code in multi-thread sequential writing.

As shown in Figs. 6 and 7, AVX2:CRS had the most similar performance to NONE, and RS had the worst performance. LIB was similar to AVX2:CRS in throughput, but the CPU utilization was high. In contrast, CRS was similar to AVX2:CRS in CPU utilization but had low throughput. In other words, the Parity Calculation accounted for 4–26% of the total IO.

In this section, we discussed the measurements of the sequential write performance of various erasure codes on real storage with single-thread and multi-thread. This experiment revealed that the EC encoding reduced the throughput by up to 30% and increased the CPU utilization by up to 20%. However, we have confirmed that using AVX2:CRS enables the storage system to perform EC with no significant performance degradation. The performance degradation because of Parity Calculation was notably small (approximately 4%). In this study, by analyzing the writing process, we identified the expected issues of EC from the exa-scale storage.

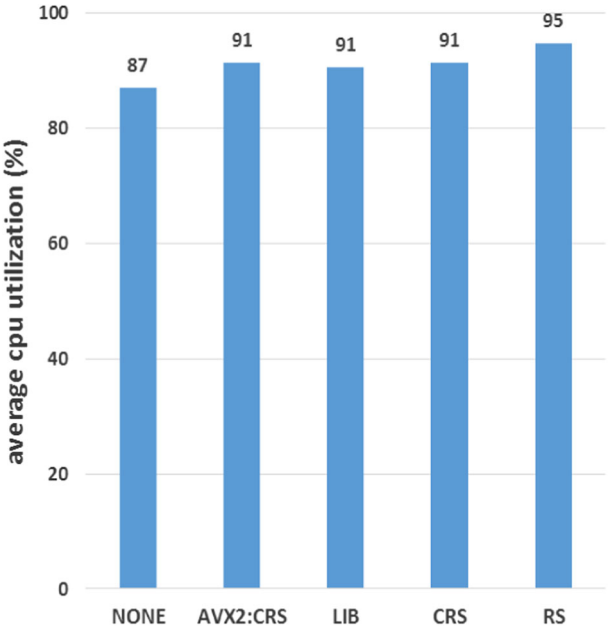

Fig. 5 Single-thread average CPU utilization by erasure code

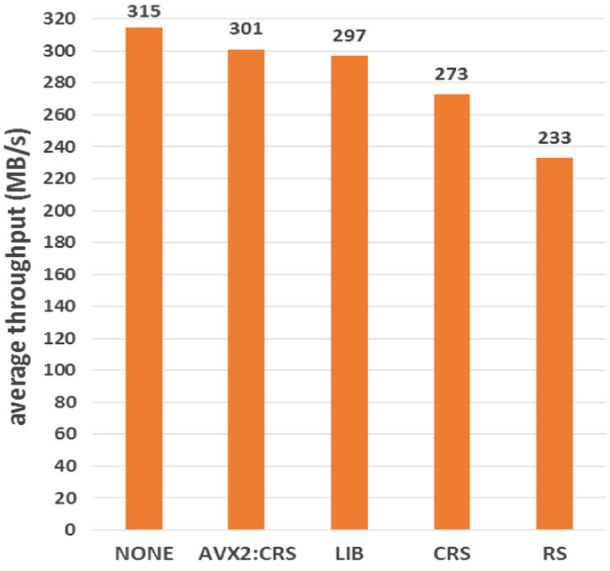

Fig. 6 Multi-thread average throughput by erasure code

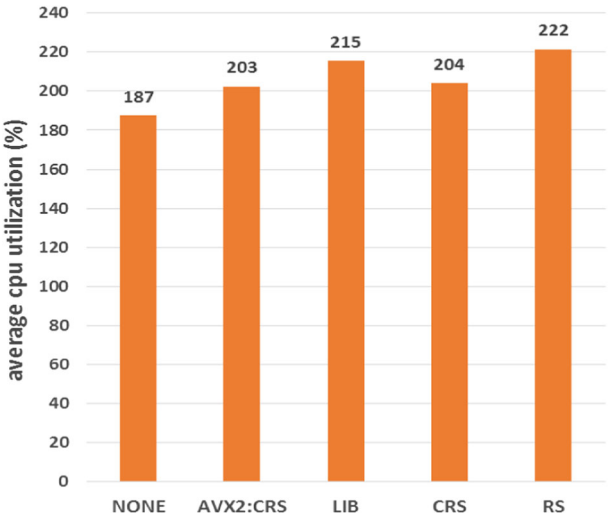

Fig. 7 Multi-thread average CPU(4-cores) utilization by erasure code

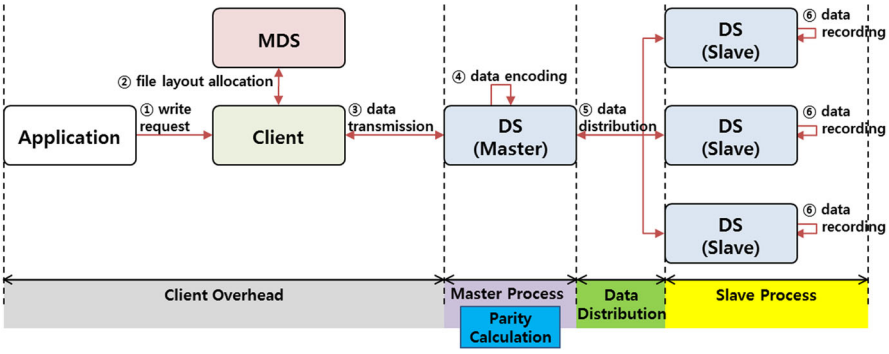

Fig. 8 Write time classification for the EC volume

3 Write cost of EC volume in exa-scale storage

In this study, we examined the effects of the Parity Calculation, Data Distribution cost, and small I/O problem among various performance degradation factors. In this section, we discuss the analysis of the write time to check the EC availability on exa-scale storage. Assuming that we use the commodity server to build exa-scale storage, we limited the change in the exa-scale storage to the bandwidth. To facilitate the analysis, we divide the write process into five steps. Figure 8 shows the write process for the EC volume.

As shown in Fig. 8, the write process was divided into five steps: Client Overhead, Master Process, Parity Calculation, Data Distribution, and Slave Process. Client Overhead excluded the time that DS processes in the entire write time. Master Process is the time excluding the Parity Calculation time, Data Distribution time, and Slave Process

time in the total DS processing time. Parity Calculation is the processing time of the EC encoding function. Slave Process was the time to receive and process data from Slave DS. Data Distribution excluded the Slave Process time from sending data to the slave and receiving the processing result.

We made several assumptions for deriving the formula to calculate the write time. First, the write was a single-thread sequential write. Second, the record size of I/O was fixed to R. Finally, the Master DS that performed the encoding was fixed. The write time (WT) is the sum of the times for each I/O processing step. Therefore, the time to write n records can be expressed as follows:

$$\mathrm{WT} = \sum_{r=1}^{n} (\mathrm{CO}_r + \mathrm{MP}_r + \mathrm{PC}_r + \mathrm{DD}_r + \mathrm{SP}_r) \tag{1}$$

where CO_r is the Client Overhead time when processing the specified records, MP_r is the Master Process time, PC_r is the Parity Calculation time, DD_r is the Data Distribution time, and SP_r is the Slave Process time.

The Client Overhead was the processing of the client, which performed several tasks such as I/O request processing, chunk allocation, layout gathering, layout analysis, file status check, data transfer, processing result gathering, file metadata update, etc. These tasks were roughly divided into three areas depending on the processing subject. First, the Client Tasks to be processed inside the client, such as the I/O request processing, layout analysis, and file status check; second, MDS Tasks to be processed through communication with MDS such as chunk allocation, layout gathering, and file metadata update; and third, DS Tasks that are processed through communication with the DS such as data transfer and processing result gathering.

Therefore, the Client Overhead could be divided into Client Tasks, MDS Tasks, and DS Tasks. CO was sum of the Client Tasks time (CT), MDS Tasks time (MT), and DS Tasks time (DT). Therefore, CO_r could be expressed as follows:

$$\mathrm{CO}_r = \mathrm{CT}_r + \mathrm{MT}_r + \mathrm{DT}_r \tag{2}$$

where CT_r refers to the internal processes without network communication, which can be replaced by a constant value according to the system specifications. MT_r is a small message-based network communication that will cost less than data transmission. Therefore, MT_r can be replaced by a constant value according to the system specifications. DT_r is defined as the latency according to record size R for the time to transmit data and wait for results. Thus, DT_r is expressed using formula (3), where LM_R is the transfer latency of record R between Client and Master. And r is a record

$$\mathrm{DT}_r = r * \mathrm{LM}_R. \tag{3}$$

The Master Process is the time to perform Master processing tasks such as chunk-set locking, memory allocation, memory copy, and encoding function call in Master. Parity Calculation is the time when the Master performs data encoding according to the erasure code. Slave Process is the time to perform slave processing tasks such as

data recording, logging and I/O locking in the slave. As MP_r, PC_r, and SP_r are internal processes without network communication, they can be replaced with a constant value according to the system specifications.

Data Distribution is the time required to transfer data from the Master to the slave and collect the processing result from the slave. Thus, DD_r is defined as the latency time to distribute data from the Master to the slave and collect the processing results from the slave. In particular, DD_r transfers the data to $K + M - 1$ Slaves except the Master and waits for all the Slaves to complete processing. Therefore, DD_r is the maximum processing time of each slave. As record R is divided into K parts according to the characteristics of the EC, the size of the data block transmitted to each slave is R/K. Hence, DD_r can be expressed as follows:

$$\mathrm{DD}_r = r * \mathrm{MAX}\left(s_1 * \mathrm{LS}\left(R/K\right), s_2 * \mathrm{LS}\left(R/K\right), \ldots, s_{K+M-1} * \mathrm{LS}\left(R/K\right) \right) \quad (4)$$

where $\mathrm{LS}\left(R/K\right)$ is the transfer latency of record R/K between Master and Slave. Then, the write time of the EC volumes in the storage is can be expressed as follows:

$$\mathrm{WT} = \sum_{r=1}^{n} (\mathrm{CT}_r + \mathrm{MT}_r + r * \mathrm{LM}_r) + \mathrm{MP}_r + \mathrm{PC}_r + r * \mathrm{MAX}\left(s_1 * \mathrm{LS}\left(R/K\right), s_2 * \mathrm{LS}\left(R/K\right), \ldots, s_{K+M-1} * \mathrm{LS}\left(R/K\right) \right) + \mathrm{SP}_r \quad (5)$$

where CT_r, MP_r, PC_r, and SP_r are constant values according to the system specifications, so $(\mathrm{CT}_r + \mathrm{MP}_r + \mathrm{PC}_r + \mathrm{SP}_r)$ is converted into a specific constant value V_r. Assuming that all the Slaves have identical latencies in DD_r, we obtained WT as follows:

$$\mathrm{WT} = \sum_{r=1}^{n} \left(V_r + \mathrm{MT}_r + (r * \mathrm{LM}_R) + \left(r * \mathrm{LS}\left(R/K\right) \right) \right) \quad (6)$$

According to formula (6), WT is affected by V_r, MT_r, LM_R, and $\mathrm{LS}\left(R/K\right)$. In general, the distributed storage occupies a large portion of the network communication in I/O. In particular, if K or M increases in EC, $\mathrm{LS}\left(R/K\right)$ increases because the number of Slaves increases and the I/O size decreases. However, if there is a high throughput as in the case of exa-scale storage, the proportion of network communication can be reduced. In other words, LM_R and $LS\left(R/K\right)$ may become smaller when the network speed increases, which may result in a relatively larger proportion of V_r and MT_r.

Table 1 Average sequential write time per processing step on 1 G Ethernet

Processing step	I/O size				
	4 K (s)	16 K (s)	32 K (s)	64 K (s)	128 K (s)
Client Overhead	1.12	1.95	2.85	3.61	5.19
Master Process	0.13	0.23	0.42	0.75	2.12
Parity Calculation	0.09	0.20	0.36	0.68	1.59
Data Distribution	16.26	28.95	53.89	91.83	153.61
Slave Process	0.61	0.95	1.35	1.95	2.57

Therefore, in Sect. 4, we discuss the experimental verification of the weight of each write step and the prediction of future problems.

4 Performance evaluations

In this section, we discuss the validation of the effectiveness of EC in the storage by measuring the performance on the real EC-based storage. In particular, we compared the performance in 1 G/10 G/40 G Ethernet to predict the performance in exa-scale storage. We will look into possible problems in the future.

This experiment measured the running time per processing step in the 4 + 2 EC volume of ExSTorus-FS. The EC volume had a 6 KB stripe, a 64 MB chunk, and the AVX2:CRS erasure code. These experiments were performed with 6 DSs, 1 MDS, and 1 client in 1 G/10 G/40 G Ethernet. The experiment was conducted using seven servers, which is the minimum number of required nodes to minimize the performance interference when measuring the performance in 4 + 2 EC by encoding the processing stage. For the performance comparison, we fixed the Master that performed the encoding.

4.1 Sequential write performance on 1 G Ethernet

In this section, we describe the measured write time for the EC volumes in a real distributed storage with 1 G Ethernet. The writing time was divided into five steps as classified in Sect. 3. In this experiment, the dd command was executed, while the I/O size changed from 4 to 128 K, e.g., dd if = /dev/urandom of = /exa/file1 bs = 4 K count = 50,000. The write time was the average value of 10 executions of the dd command. Table 1 shows the average sequential write times per processing step with 1 G Ethernet.

Figure 9 shows the execution time in Table 1 expressed in unit time divided by the number of write times (50,000).

As shown in Fig. 9, all the unit times were proportional to the I/O size. The Client Overhead and Slave Process were approximately four times the difference between 4 and 128 K. Master Process and Parity Calculation were approximately 17 times the

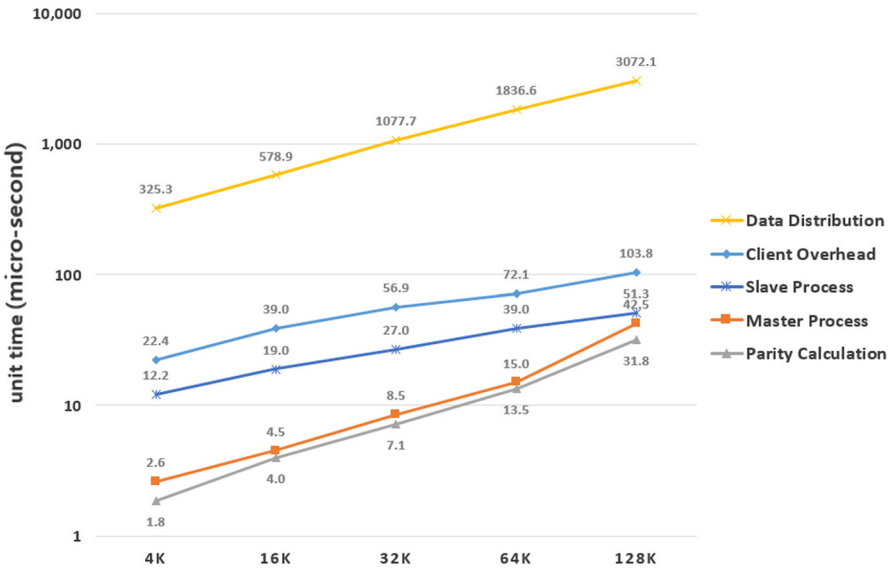

Fig. 9 Unit time per processing step in 1 G Ethernet

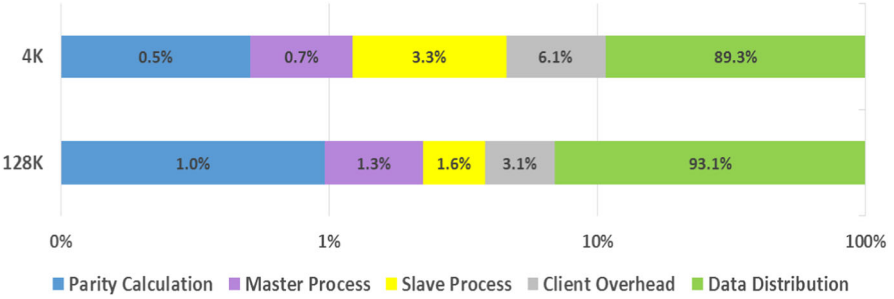

Fig. 10 Percentage of execution per processing step in 1 G Ethernet

difference between 4 and 128 K. Data Distribution was approximately nine times the difference between 4 and 128 K.

The Data Distribution, Client Overhead, and Slave Process were performed in the record, whereas the Master Process and Parity Calculation were performed in the stripe. Therefore, the Master Process and Parity Calculation significantly increased the execution time because the number of operations increases when the I/O size increased. In addition, the Data Distribution increased the processing time because the I/O size increased and the transmission time increased. Because the Client Overhead operated in the write-back mode, the time did not significantly increase even when the I/O size increased.

Figure 10 shows the percentage of execution time per processing step at 4 K and 128 K for bottleneck identification.

Table 2 Average sequential write time per processing step on 10 G Ethernet

Processing step	I/O size				
	4 K (s)	16 K (s)	32 K (s)	64 K (s)	128 K (s)
Client Overhead	1.28	2.67	3.4	4.56	6.28
Master Process	0.15	0.38	0.59	0.92	2.05
Parity Calculation	0.09	0.23	0.39	0.69	1.46
Data Distribution	7.61	14.59	16.24	17.98	20.73
Slave Process	0.53	1.05	1.34	1.70	2.33

As shown in Fig. 10, the proportion of Data Distribution was the highest at approximately 93.1%. The proportion of Client Overhead was the second highest at approximately 3.1%. The proportion of Parity Calculation was the lowest at approximately 1%. With 1 G Ethernet, the transmission speed was notably low, so the Data Distribution took up most of the total execution time. The Parity Calculation was sufficiently low to be ignored.

4.2 Sequential write performance on 10 G Ethernet

In this section, we describe the measured write time for the EC volumes in real distributed storage with 10 G Ethernet. The write time was divided into five steps as classified in Sect. 3. In this experiment, the dd command was executed, while the I/O size changed from 4 to 128 K, e.g., dd if=/dev/urandom of=/exa/file1 bs=4 K count=50,000. The write time was the average value after 10 executions of the dd command. Table 2 shows the average sequential write times per processing step with 10 G Ethernet.

Figure 11 shows the execution time in Table 2 expressed in unit time divided by the number of write times (50,000).

As shown in Fig. 11, all the unit times were proportional to the I/O size. The Client Overhead and Slave Process were approximately 4.5 times the difference between 4 and 128 K. The Master Process and Parity Calculation were approximately 13 times the difference bet ween 4 K and 128 K. The Data Distribution was approximately three times the difference between 4 and 128 K.

Figure 12 shows the percentage of execution time per processing step at 4 K and 128 K to identify the bottlenecks.

As shown in Fig. 12, the proportion of Data Distribution increased to 78.7% with a smaller I/O size. The proportion of the Client Overhead, Slave Process, Master Process, and Parity Calculation increases when the I/O size increased. The Client Overhead was the second highest with a maximum of approximately 19.2%. The Parity Calculation was the smallest with a maximum of approximately 4.1%. When the network speed increased, the Data Distribution decreased, and the Client Overhead, Slave Process, Master Process, and Parity Calculation increase.

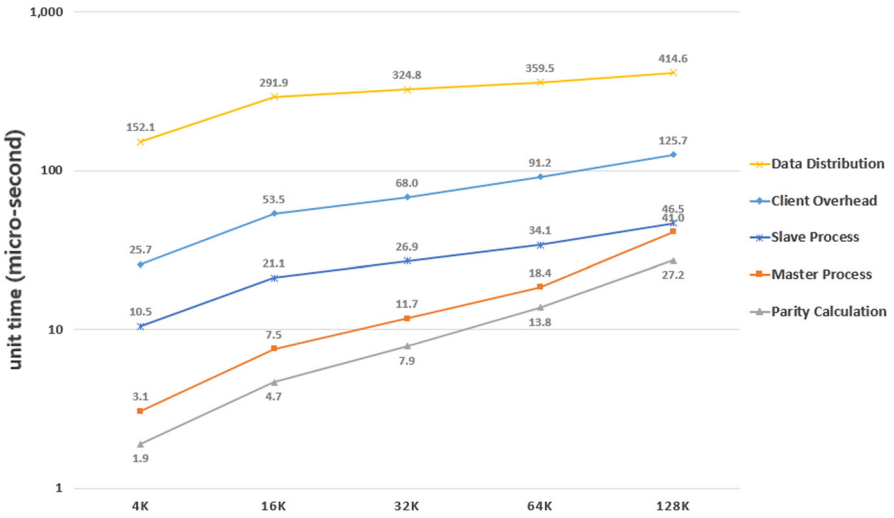

Fig. 11 Percentage of execution per processing step in 10 G Ethernet

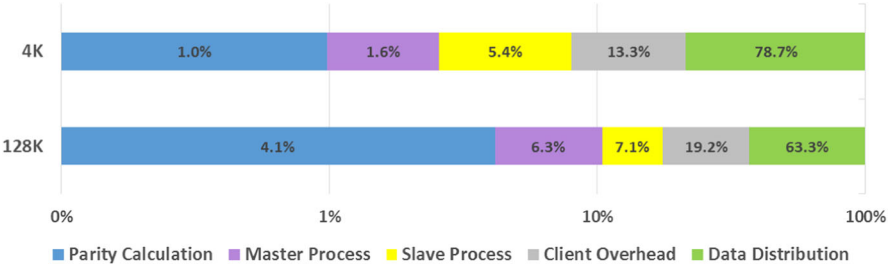

Fig. 12 Percentage of execution per processing step in 10 G Ethernet

Figure 13 shows the execution time for each step of 1 G and 10 G for 128 K I/O, which had the longest encoding time.

As shown in Fig. 13, the Slave Process, Master Process, and Parity Calculation, which were not affected by the network speed, had similar performances between 1 and 10 G. However, the performance of the Client Overhead and Data Distribution, which were affected by the network speed, varied between 1 and 10 G. In particular, the time for the Data Distribution decreased with an increase in the network speed from 1 to 10 G, whereas the time for Client Overhead increased.

4.3 Prediction of write time of EC in exa-scale storage

In this section, we discuss the estimation of the write cost on exa-scale storage based on the experimental results presented in Sects. 4.1 and 4.2. The Master Process, Parity Calculation, and Slave Process were the same irrespective of the network performance in EC performance result. Data Distribution and Client Overhead were related to the network throughput. In particular, exa-scale storage increased the throughput, capacity,

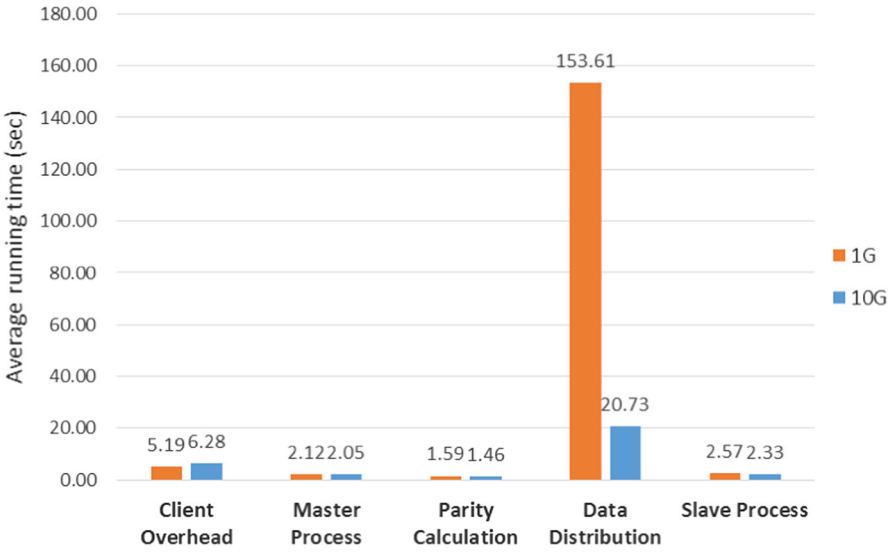

Fig. 13 Comparison of processing steps of 1 G and 10 G

Table 3 Average 128 K sequential write time per processing step

Processing step	Ethernet		
	1 G (s)	10 G (s)	40 G (s)
Client Overhead	5.19	6.28	6.68
Master Process	2.12	2.05	2.15
Parity Calculation	1.59	1.46	1.64
Data Distribution	153.61	20.73	11.82
Slave Process	2.57	2.33	2.48

system size, power usage, number of files, frequency of failure, etc., in the range of tens to hundreds of times as compared to the existing storage. Among these, network performance was the most important factor affecting the EC performance.

We performed additional experiments on 40 G in system environments identical to those described as Sects. 4.1 and 4.2 to increase the reliability of the write cost estimation results in exa-scale storage. Table 3 shows the average sequential write time of each processing step for 1 G/10 G/40 G for the 128 K I/O with the longest encoding time.

Moreover, as shown in Fig. 13, we assumed that Master Process, Parity Calculation, and Slave Process were the same irrespective of the network performance. Data Distribution and Client Overhead were related to network throughput. However, as Table 3 shows, the Data Distribution decreased when the network speed increased, but the Client Overhead increased as network speed increased.

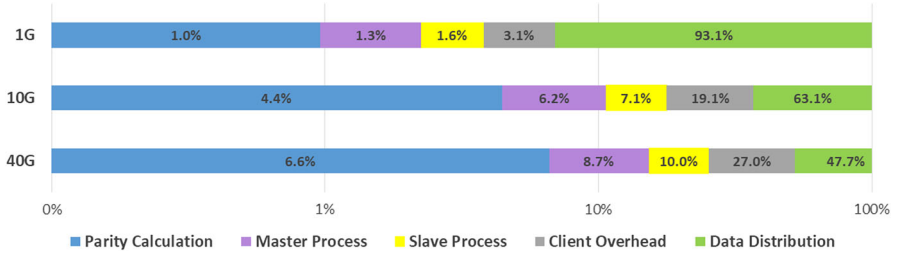

Fig. 14 Percentage of execution per processing step in 1 G/10 G/40 G

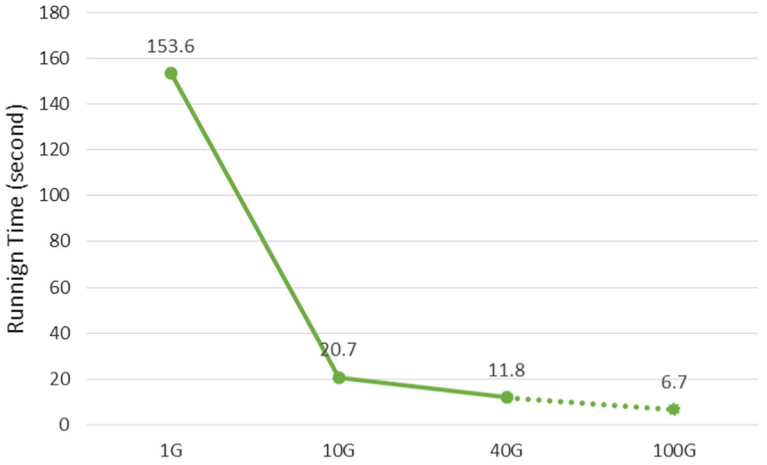

Fig. 15 Prediction of Data Distribution time

Figure 14 shows the percentage of execution time per processing step at 128 K to identify the bottlenecks. Figure 15 shows the performance graphs of Data Distribution predicted based on the ratio of 1 G, 10 G, and 40 G.

As shown in Fig. 14, the proportion of Data Distribution reduced to 47.7% with an increase in the network speed. The proportion of the Client Overhead, Slave Process, Master Process, and Parity Calculation increased when the network speed increased. The Client Overhead was the second highest with a maximum of approximately 27%. The Parity Calculation was the smallest with a maximum of approximately 6.6%.

We assumed that a commodity server and 100 G were used in the exa-scale storage for write cost prediction. As shown in the experimental results presented in Table 3 and Fig. 14, Master Process, Parity Calculation, and Slave Process in the 1 G/10 G/40 G were almost the same irrespective of the network performance. Moreover, the Data Distribution and Client Overhead were related to the network throughput. The Client Overhead was similar to 40 G because the performance difference was not large. However, Data Distribution varied considerably in performance depending on the network throughput. Figure 15 shows the predicted Data Distribution time.

As shown in Fig. 15, the performance ratio of 1 G, 10 G, and 40 G was applied to 100 G to predict the execution time of Data Distribution.

Fig. 16 Prediction of write time of EC in exa-scale storage

Figure 16 shows the percentage of execution time per processing step for the bottleneck identification.

As shown in Fig. 16, assuming 100 G, we found that the Data Distribution was reduced to approximately 17%, but the Parity Calculation, Master Process, Slave Process, and Client Overhead relatively increased. In particular, Client Overhead increased by approximately 46% at 100 G. The proportion of Data Distribution sharply decreased from 93.1 to 17.2% but remained the largest. The proportion of Parity Calculation sharply increased from 1 to 9.2%, but it was still the smallest.

More importantly, Client Overhead rapidly increased by 15 times. Assuming that Client Overhead increases at 100 G as well as at 40 G, we found that the proportion of Client Overhead increased sharply. As described in formula (2), the Client Overhead was composed of CT, MT, and DT. If CT was similar to Master Process and DT was reduced like Data Distribution, MT increased rapidly.

5 Conclusions

In this study, we measured the various I/O performances of erasure codes when EC was used in storage, and we confirmed that encoding could be performed with small performance degradation when the SIMD was used. In addition, the network transmission cost was the largest performance hurdle factor, and the proportion of the encoding cost was not sufficiently large for a cost analysis of the EC.

Finally, we found that the Client Overhead could be a major problem in the future through EC cost estimates based on the bandwidth of the exa-scale storage. In detail, assuming 100 G, we found that the cost of the Client Overhead increased by approximately 46%, and the client process time was two times larger than the Data Distribution time. In the future, it will be necessary to more specifically analyze the Client Overhead area and identify the bottlenecks.

Acknowledgements This work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIT) (No.2015-0-00262, Management of Developing ICBMS(IoT, Cloud, Bigdata, Mobile, Security) Core Technologies and Development of exa-scale Cloud Storage Technology).

References

1. Apache Hadoop 3.0.0 (2017) https://hadoop.apache.org/
2. Ceph (2016) http://docs.ceph.com/docs/master/rados/
3. Chu X, Liu C, Ouyang K, Yung LS, Liu H, Leung YW (2015) PErasure: a parallel Cauchy Reed—Solomon coding library for GPUs. In: Proceedings of the 2015 IEEE International Conference on Communications (ICC), London
4. Cook JD, Robert P, Kwant A (2014) Compare cost and performance of replication and erasure coding. Hitachi Review. http://www.hitachi.com/rev/pdf/2014/r2014_july.pdf
5. Dustin B (2017) Red hat gluster storage on supermicro storage servers powered by intel xeon processors. Super Micro computer. https://www.supermicro.com/white_paper/white_paper_Redhat_Gluster_Storage.pdf
6. Filesystem in Userspace (2015) https://sourceforge.net/projects/fuse
7. Glusterfs (2018) https://access.redhat.com/products/red-hat-storage/
8. Haddock W, Curry ML, Bangalore PV, Skjellum A (2017) GPU erasure coding for campaign storage. In: Proceedings of the International Conference on High Performance Computing. ISC High Performance 2017, Frankfurt
9. Huining Y, Yiming Z, Huaimin W, Bo D, Haibo M (2017) S3R: storage-sensitive services redeployment in the cloud. Int J Big Data Intell 4(4):250–262
10. Hsing-bung C, Song F (2016) Parallel erasure coding: exploring task parallelism in erasure coding for enhanced bandwidth and energy efficiency. In: Proceedings of the 2016 IEEE International Conference of Networking, Architecture and Storage (NAS), California
11. Intel Intelligent Storage Acceleration Library (Intel ISA-L) Open Source Version, API Reference Manual-Version 2.10 (2014) https://01.org/sites/default/files/documentation/isa-l_open_src_2.10.pdf
12. Intel Storage Acceleration Library (2017) https://software.intel.com/en-us/storage/ISA-L
13. Iozone Filesystem Benchmark (2017) http://www.iozone.org/docs/IOzone_msword_98.pdf
14. Jun L, Baochun L (2013) Erasure Coding for cloud storage systems: a survey. Tsinghua Sci Technol 18(13):259–272
15. Jun L, Mengshu H (2018) Improving data availability for deduplication in cloud storage. Int J Grid High Perform Comput 10(2):70–89
16. Khan O, Burns RC, Plank JS, Pierce W, Huang C (2012) Rethinking erasure codes for cloud file systems: minimizing I/O for recovery and degraded reads. In: Proceedings of the 10th USENIX Conference on File and Storage Technologies, California
17. Kim CY, Kim DO, Kim HY, Kim YK, Seo DW (2016) Torus network based distributed storage system for massive multimedia contents. J. Korea Multimed Soc 19(10):1487–1497
18. Kim CY, Kim YC, Kim DO, Kim HY, Kim YK, Seo DW (2016) Implementation and performance measuring of erasure coding of distributed file system. J. Korean Inst Commun Inf Sci 41(11):1515–1527
19. Kim YC, Kim DO, Kim HY, Kim YK, Choi W (2013) MAHA-FS: a distributed file system for high performance metadata processing and random IO. KIPS Trans Softw Data Eng 2(2):91–96
20. Kunkel JM, Michael K, Thomas L (2014) Exascale storage systems—an analytical study of expenses. Supercomput Front Innov 1(1):116–134
21. Mohan LJ, Harold RL, Caneleo PIS, Parampalli U, Harwood A (2015) Benchmarking the performance of hadoop triple replication and erasure coding on a nation-wide distributed cloud. In: Proceedings of the 2015 International Symposium on Network Coding (NetCod), Sydney
22. Nicloe H (2017) An exascale timeline for storage and I/O systems. The Next Platform. https://www.nextplatform.com/2017/08/16/exascale-timeline-storage-io-systems/
23. Peter S (2012) Parallel coding for storage systems—an OpenMP and OpenCL capable framework, In: Proceedings of ARCS 2012 Workshops, Muenchen
24. Plank J. S. (1997) A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems. Softw.-Pract. Exper 27(9):995-1012
25. Plank JS (2013) Erasure codes for storage systems. The Usenix magazine. https://www.usenix.org/system/files/login/articles/10_plank-online.pdf
26. Plank JS, Scott S, Schuman CD (2008) Jerasure: a library in C/C ++ facilitating erasure coding for storage applications—version 1.2. Technical report. Department of Electrical Engineering and Computer Science University of Tennessee. https://web.eecs.utk.edu/~plank/plank/papers/CS-08-627.pdf
27. Plank J. S. (2008) The RAID-6 liberation codes. In: Proceedings of the 6th Usenix Conference on File and Storage Technologies, California

28. Plank JS, Xu L (2006) Optimizing Cauchy Reed–Solomon codes for fault-tolerant network storage applications. In: Proceedings of 2006 Fifth IEEE International Symposium on Network Computing and Applications (NCA), Massachusetts
29. Patterson David A, Hennessy John L (1998) Computer organization and design. In: The hardware/software interface (2nd ed.). Elsevier Science, ISBN 155860491X
30. Red Hat (2014) Technology detail: Red Hat Ceph storage hardware configuration guide—Designing scalable workload-optimized Ceph clusters. https://www.redhat.com/cms/managed-files/st-rhcs-config-guide-technology-detail-inc0387897-201604-en.pdf
31. Sun D, Xu Y, Li Y, Wu S, Tian C (2016) Efficient parity update for scaling RAID-like storage systems. In: Proceedings of 2016 IEEE International Conference on Networking, Architecture and Storage (NAS)
32. Takeshi M, Takanori N, Kensuke S (2014) Erasure code with shingled local parity groups for efficient recovery from multiple disk failures. In: Proceedings of the 10th USENIX Conference on Hot Topics in System Dependability, California
33. The Pros and Cons of Erasure Coding & Replication vs. RAID in. Next-Gen storage platforms (2017) http://www.snia.org/sites/default/files/SDC15_presentations/datacenter_infra/Shenoy_The_Pros_and_Cons_of_Erasure_v3-rev.pdf