iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 7 **Issue:** IV **Month of publication:** April 2019

DOI: https://doi.org/10.22214/ijraset.2019.4066

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 6.887
Volume 7 Issue IV, Apr 2019- Available at www.ijraset.com

Erasure Coding in Multi–block Updates

Dr. S. Gunasekaran[1], Mr. S. Gokul[2], Ms. A. Nivetha[3], Mr. R. Praveena[4], Mr. P. Ramar[5]
[1, 2, 3, 4, 5]*Department of CSE, Coimbatore Institute of Engineering and Technology, Coimbatore, India*

Abstract: ***The purpose of the problem is how do reduce and manage the I/O overhead of data updating process in erasure coding. Mapping the update operations with current updating process. We find the method to reduce and maintains I/O overhead designing a heuristic scheduling algorithm. Then the analysis verifies that modify UCODR can effectively reduce the I/O overhead of update operations when multiple blocks are updated. It is implement a efficient storage system to deploy a modify UCODR with different set of erasure codes. As the results UCODR can reduce the time of update operation by 37 percentage and the input throughput of storage system increased 67 percentage.***
Keyword: Erasure codes, data updates, disk array, Cloud computing, Disk I/O overhead.

I. INTRODUCTION

Erasure Coding is a one of the data protection method widely used in big data storage systems.Erasure coding break the data into small piece of fragment and encode the data fragment with redundant pieces which can be stored in different storage locations. The real goal of Erasure coding is data protection and Storage system. The data reliability of storage systems that gives guarantee and widely advocated by ERASURE codes [1]. (e.g., disk array, distributed storage systems and cloud storage) .

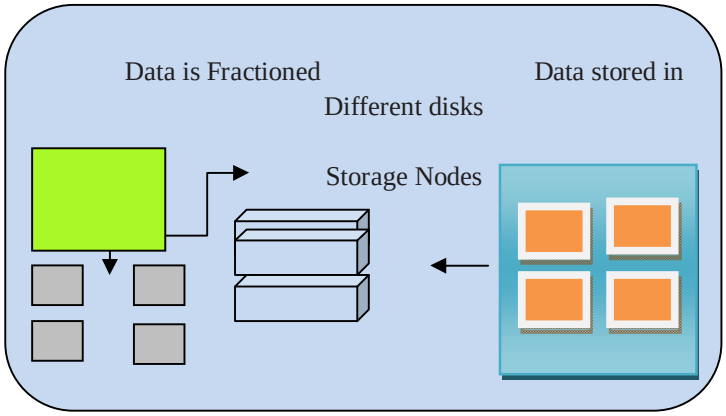


Fig.1 Erasure coding and data fractioned into disks

A. I/O during data update:

Disk I/O works on the input/output operations on a physical disk. When the processor needs to wait for the file to be read because you're reading data from a file on a disk (the same goes for writing). This is the time required for a computer to process a data request and response the processor and then retrieve the required data from the correct device. Reducing the I/O overhead of the update operations is hence a critical concern towards applying erasure codes in online applications. The update approaches only consider Small data values and Update and optimize the each block data Independent manner.

B. Latency Constraints:

It is the delay from input to a storage system to desired outcome; the term is understood the different in various process and latency issues also varied from one to another. The data recovery process, update requests generally work under a stringent latency constraint. As a result, the scheduling algorithm needs to construct update sequences within a time limit. We have to resort to a heuristic algorithm to quickly find a good update sequence that can effectively reduce the I/O overhead of update operations. System and latency is the combined delay between input and the correct output. In a computer system, often used to mean any delay or waiting that increases working the perceived response time beyond what is desired.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 6.887
Volume 7 Issue IV, Apr 2019- Available at www.ijraset.com

C. *Implement of Erasure Coding*

The method broken the some data and parity data add the different locations.

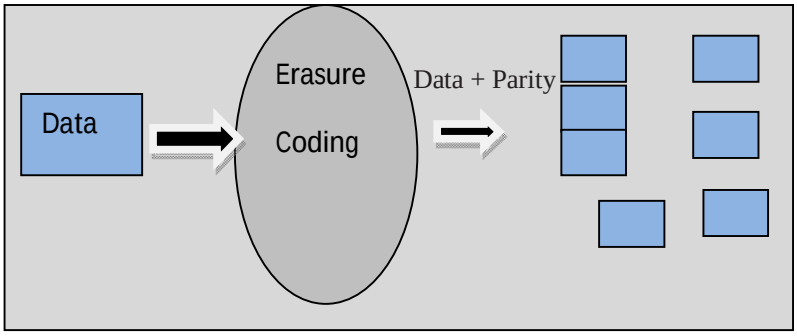


Fig.2 diagram formation of parity block

A simple example of parity calculation is as follows:

1) x = 2 (data)
2) y = 5 (data)
3) x + y = 7 (parity)
4) 2x + y = 9 (parity)

Hence, if "x=2" is lost, it can be reconstructed using the remaining equations. Even if both data bits are lost: "x=2" and "y=5", both can be reconstructed. The systems is designed of n disk. The disks are partition into k disks that hold user data so that..m=n-k disks and hold coding information.

The encoding contents of the k data disks are used to calculate the contents of the m coding disks. Their contents are decoded from the surviving disks, When up to m disks fail. Repeat all disks are in failure mode is an erasure, Then it's all content are not unreadable.

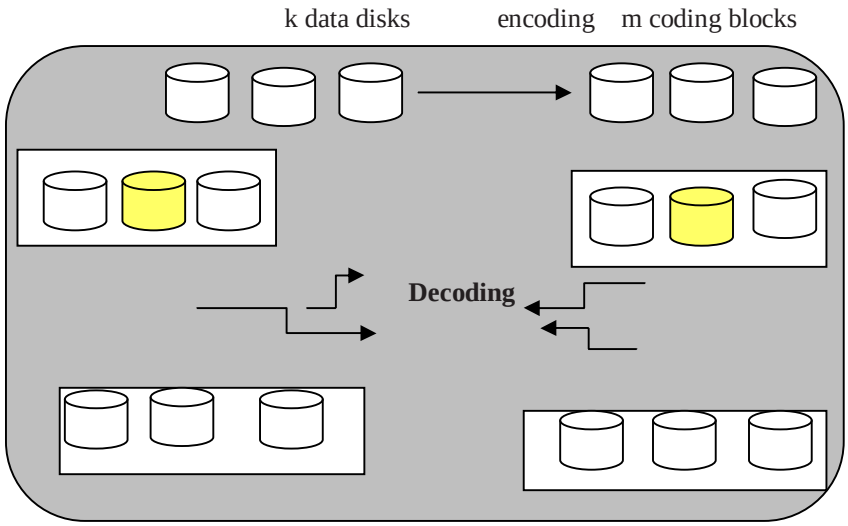


Fig.3 encoding and decoding of k disks and m disks.

The simplest erasure codes assume that each disk

Each one w-bit word. I label these words $x_0, \ldots, x_{k-1}$, which are the data words stored on the data disks, and $c_0, \ldots, y_{m-1}$, which are the coding words stored on the coding disks. The coding words are

defined as linear combinations of the data words

$$y_0 = n(_{0,0})x_0 + \cdots . + n(_{0,k-1})x_{k-1}y1 = n(_{1,0})x_0 + \cdots . + n_{(1,k-1)}x_{k-1y} \quad (1)$$

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 6.887
Volume 7 Issue IV, Apr 2019- Available at www.ijraset.com

II. REED SOLOMON CODE

Clever ways of representing data so that one can recover the original information even if it is corrupted. In reed Solomon code the data will recover using redundancy parts of corrupted.

Reed Solomon codes are error correcting codes that have been used in applications through out the fields storage systems and digital information. The Parameter of Reed-Solomon codes n (block size), m (message size), p (symbol size in bits), we encode the message as a polynomial p(x), and then multiply with a code generator polynomial r(x).We construct code generator polynomial r(x) with n –k factors, each root being a consecutive element in the Galois field.

$r(x) = (x-\alpha)(x-\alpha^2)\ldots\ldots(x-\alpha^{n-m}) = r_0 + r_{1x} + \cdots . + r_{n-m-1}x^{n-m-1} + x^{n-m}$ (2α is a primitive element, an alternative way of specifying elements in a field as successive powers 0, α0, α1, α2... αN where N = 2p-1.

Set the code parameters.

p=Number of bits per symbol

n=Codeword length

m=Message length

$p = 3$

$n = 2^m - 1$ (3)

$m = 3$

Create two messages based on GF(8).

message = gf([2 7 3; 4 0 6],m)

message = GF(2^3) array. Primitive polynomial = D^3+D+1 (11 decimal)

Array elements = $\begin{pmatrix} 2 & 7 & 3 \\ 4 & 0 & 6 \end{pmatrix}$

Generate RS (7,3) code words.

code = rsenc (message ,n, m)

code = GF(2^3) array. Primitive polynomial = D^3+D+1 (11 decimal)

Array elements =

$$\begin{bmatrix} 2 & 7 & 3 & 6 & 7 & 6 & 6 \\ 4 & 0 & 6 & 4 & 2 & 2 & 0 \end{bmatrix}$$

the message has been written in vector $[X_1 \ldots X_n]$ to a polynomial p(x) of degree < k such that:•

$p_x(\alpha_i) = x_i \; holds \; that \; i = 1\ldots\ldots k$ (4)

Can be done using Lagrange interpolation. Once polynomial is found, evaluate at other for other points αk+1 ... n Nice property that evaluating p at the first k points yields the original message symbols.

Decoding: The received block is input to the decoder for processing, the decoder first verifies whether this block appears in the dictionary of valid code words. Errors must have occurred during transmission when it does not have anything. This part of the decoder processing is called error detection .for a decoder, The parameters must to reconstruct the encoded block are available . If all the errors detected, the decoder attempts a reconstruction. This is called error correction.

Syndromes calculation: the received block, the received polynomial is reconstructed, represented as c(x). The received polynomial is the superposition of the correct code word c(x) and an error polynomial e(x):

$r(x) = c(x) + e(x)$ (5)

Since c(x) is multiple of r(x)

$c(x) = p(x)r(x)$ (6)

A. Goals

Always on erasure coding, drive failure tolerance, low space overhead, high performance.

The erasure codes enable that the corrupted data at some point in the drive storage process to reprocessed by using information that data will e stored in array.

Parity blocks: means that is a technical process it checks the data has been lost or written over then its moved to one place in storage to another or it could be transmitted to another place. How it works, the process of parity block first checks the adds checksums into data that enable the target device to determine whether the data was received properly.

Even parity, a value of one are counted. perhaps that number is odd with the number of bits, the parity bit value is to make the total number of ones an even number. All the number of bits with a value of one is even, the parity bit value is zero, so that the total of ones in the set include the all remains an even number.

odd parity: the number of bits with a value one is an even number, the parity bit value is set to make the total number of ones in the set contains the parity bit an odd number. the number of bits with a value is odd, the parity bit value is zero, so that the total number of ones in the set (contains the parity bit) remains an odd number.

III. PROPOSED WORK

A. File Update Approach

Discuss, how the update requests execute in storage systems. we should show how the update requests are mapped into storage systems, and then define two approaches to update parity blocks RCW and RMW are two elements to compare the I/O overhead of update approaches. Finally, we propose a direct update approach, namely data block update. The summarization and notation of the equations are completed represented as follows.

B. Mapping of Update Requests

The update process of storage systems and update requests, we first derive how storage systems map update requests into the data and parity blocks. Erasure codes typically portioned into multiple stripes , we now use an example to introduce update request which mapped into the stripe. The update request of the i^{th} file is mapped into three data blocks d0, d1, d2 and parity block p0. We can calculate parity block p0 by the following expression

$$h0 = w0 \oplus w1 \oplus w2 \ldots . \oplus w_{k-1,} \qquad (7)$$

$w_i$is represented as the i^{th} data block in a given stripe or data block set. h_i is represented as the i^{th}parity block in a given stripe or parity block set. $w_{i,j}$ represented as the j^{th} data block in the i^{th} data disk in a given stripe.

Where $\oplus$ represents the XOR operation. For simplicity, we use $\oplus_{i=0}^{l} w_i$ to represent $w0 \oplus w1 \oplus w2 \ldots . \oplus w_i$Therefore, we can transform Eq. (1) to

$$h0 = \oplus_{i=0}^{k=1} wi \qquad (8)$$

w_i^r is represented as the value of data block wi in update round r.$h_{i,j}$ represented as the j^{th} parity block in the i^{th} parity disk in a given stripe .Since the update request is mapped into the stripe, it incurs the data changes in both data blocks and parity blocks. As a result, the storage system should update several data blocks, as well as parity blocks, to maintain the data consistency of the stripes. Next, we define two approaches to update parity blocks according to update requests.

C. Two Kinds of Update Approaches

There are two types of blocks when storage systems update data. One type of blocks will maintain their original values after storage

1) *Reconstructed-Write (RCW)*: The purpose of RCW its creates new data block from Existing blocks. The following expression for RCW

$$h_i^{r+1} = \left(\oplus_{j=0}^{v-1} u_j^{r+1}\right) \oplus \left(\oplus_{l=0}^{y-1} o_l^{r+1}\right) \quad (9)$$

h_i^r represented as the value of parity block hi in update round r. u_i^r represented as the value of the i^{th}updated data block in update round r.

o_i^r represented as the value of the i^{th} original data block in update round r.RCW reads original data blocks and reconstructs the new value p_i^{r+1} i of the parity block p_i in update round r +1.Where u_j^{r+1} is the value of updated data block uj ando_l^{r+1} l is the value of original data block o_l in update round r +1.

2) *Read-Modify-Write (RMW):* The purpose of the RMW reads the original content and parity blocks modify old values. The following expressions for RMW

$$h_i^{r+1} = h_i^r \oplus \left(\oplus_{j=0}^{v-1} u_j^{r+1}\right) \left(\oplus_{j=0}^{v-1} u_j^r\right) \quad (10)$$

u_i^r represented as the value of the i^{th}updated data block in update round r.o_i^r represented as the value of the i^{th} original data block in update round r

RMW reads original data blocks and parity blocks modifies the old value h_i^r to the new value h_i^{r+1} i in updated round r+1 . The system finished the update process. The other will modify their data according to update requests. We call them original blocks and updated blocks, respectively. Suppose there are y original data blocks and v updated data blocks in the stripe. parity block pi has been updated for r rounds, we use p_i^r to represent the value of parity block pi in update round r. There are two approaches to update parity block pi [15].

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 6.887
Volume 7 Issue IV, Apr 2019- Available at www.ijraset.com

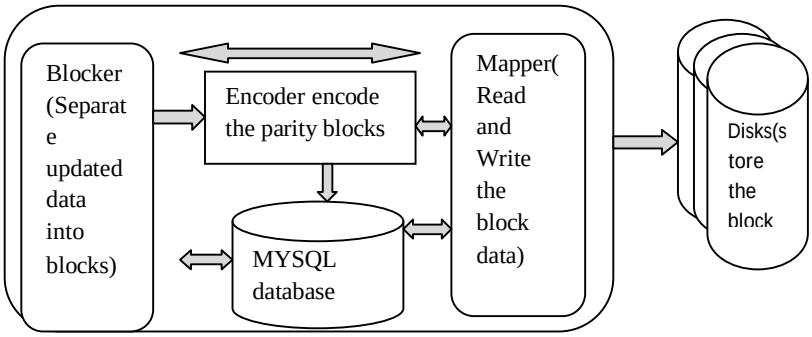


Fig.4. The software architecture of SWALLOWFS.

IV. MODIFIED UCODR ALGORITHM

Modified UCODR algorithm has two update approaches(RCW,RMW).Accordingly we construct the scheduling algorithm modified as UCODR. In real-world challenges, Analysis of how the update sequence are reason for affection in I/O overhead. Based on the notion, we propose an algorithm modified UCODR scheduling algorithm. Through theoretical analysis ,and we prove that Modified UCODR algorithm can effectively reduce the I/O overhead of update operations in practical storage systems. This algorithm implementation is also connected with Big data storage systems.

Scheduling algorithm is update sequence with in time limit. When updating the parity blocks ,Modified UCODR minimizes the I/O overhead which caused by reading the data blocks. The parity blocks are encoded with XOR data blocks. We implement a prototype storage system, namely SWALLOWFS. The SWALLOWFS contains about 45,000 lines of codes in C language. In which the source code available at

https://githud.com/UCODR/SWALLOWFS .

the storage system executes updates operation as follows. The storage system update data blocks according to the update requests. a blocker to construct data block by the implementation. the storage system uses scheduling algorithm to encode parity blocks to data blocks. Accordingly ,We use MYSQL database to store metadata files including like(e.g, filename and attributes)

UCODR can use either data or parity blocks as source blocks to update each parity block.

Accordingly UCODR needs to compare the elements of both data and parity blocks as follows.

If the source element is a data block, UCODR compares. The read data block number of RCW for parity block p_i and the read data block number of RMW for parity block p_i.

$$C_{total} = C_{data} + C_{data} \leq 4(w.m - 1) + 2 = 4.w - 2 \qquad (11)$$

V. EVALUATION

To verify the efficiency of our approach. We should introduce environment setting and then, analyze the experimental results to show the advantage of our scheme compared with current update approaches.

A. Environment Setup:

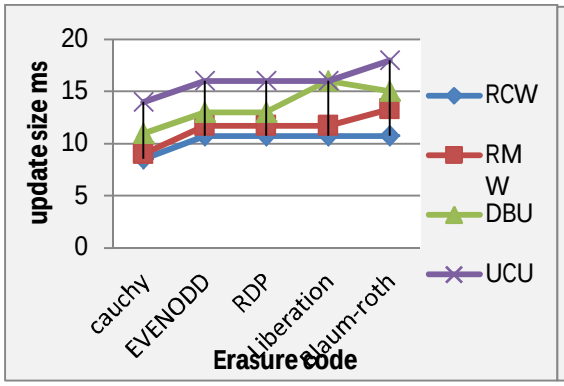
We deploy the prototype storage system in a HP Pro book 5320m server with Seagate HDD (Hard Disk Drive) disks. We mount these disks in the local file system, so that all the disk I/O operations can be executed within the user application. To better simulate the real data storage environment, we employ a real-world I/O level trace set. In particular, We only focus on the write requests, since the read requests do not involve the update process. Note, the read workload can enhance the efficiency of update operations by caching the blocks, which have been read by the read workload, in the memory. Since various erasure codes are used in practice, we also employ five erasure codes. Such codes are widely adopted in reality to ensure data reliability [2], [3], [4], [5], [6], [7], [8]. The setting of these erasure codes is depicted .

Evaluate update time (i.e., the time to complete the update process) and throughput, which is measured by the processed data volume divided by the cumulative update time. These metrics are widely used in evaluating the performance of storage systems [12], [13], [14], [15], [16], [17]. For each trace, we randomly select 100 continuous write requests, and let our prototype storage system perform such requests. Each result is obtained by running the experiment for 1000 times, respectively. Average the results to obtain the performance of the setting and then use ms (millisecond) and MB/s (MB per second) as units to measure update time and through put. We compare our proposed scheme UCU with current update approaches (i.e., RCW, RMW, and DBU) under different traces, update sizes, block sizes, and erasure codes, since these approaches are widely used to update parity blocks in practice [14],

[15], [16], [17]. In particular, UCU can reduce the update time by up to 35 percent compared with RCW, 26 percent compared with RMW, and 15 percent compared with DBU, and improve the throughput by up to 67 percent compared with RCW, 42 percent compared with RMW, and 27 percent compared with DBU. Moreover, we also evaluate the performance of update operations for 5 different block sizes: 16 KB, 32 KB, 64 KB, 128 KB, and 256 KB, since the default block sizes of storage systems are typically smaller than 256 KB [21], [22], [23]. The results are shown in Figs. 12b and 13b. In general, UCU can efficiently enhance the efficiency of data update compared with current approaches when the block size is small, since more data blocks will be updated with the same update request. Then, we compare the efficiency of update operations under 5 different update sizes: 32 KB, 64 KB, 128 KB, 256 KB, and 512 KB, since the update sizes are generally smaller than 512 KB in practice [14]. The results are shown in Figs. 5 and 6. In short, UCU can significantly improve the performance of storage systems
compared with current update approaches under different update sizes. The results are shown in
Figs. 5 and 6. For different erasure codes, UCU can improve the performance of update operations compared with other approaches. Therefore, our scheme can effectively enhance the efficiency of data update in reality, since these codes are widely deployed in practice.

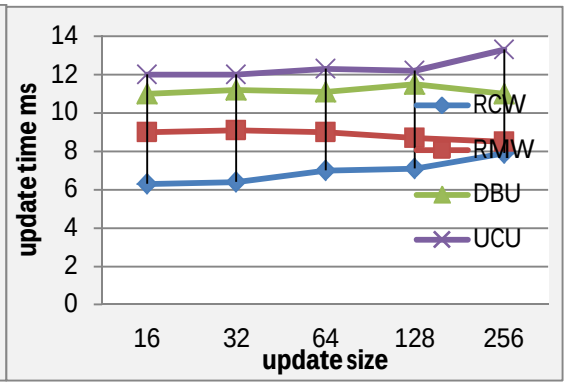
VI. CONCLUSION

The disk I/O overhead of data updates is one of the most important bottlenecks in modern storage systems. In this paper, we consider the problem of how to reduce the I/O overhead for multi-block updates in erasure coding based storage systems. By analyzing the update process, we figure out that the I/O overhead of update operations is severely affected by the order of these operations. Accordingly, we propose an efficient algorithm, namely UCODR, to reduce the I/O overhead by scheduling the update sequence. We further evaluate the performance of UCODR in a prototype storage system with real world traces. The experimental results show that UCODR can significantly improve the performance of update operations for different erasure codes, compared with current update approaches. In future work, we plan to deploy UCODR in a real-life storage system.



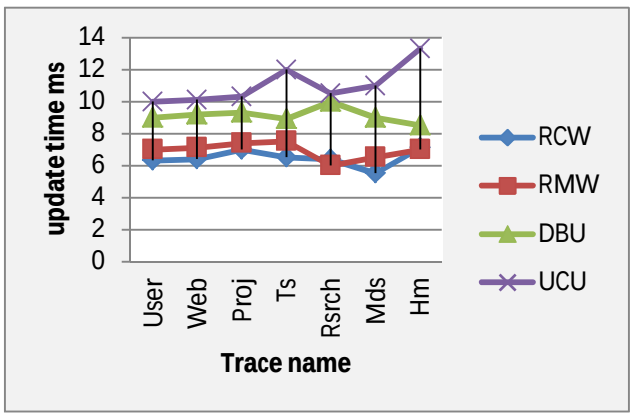
ERASURE CODE

a).Update time for RCW,RMW,DBU UCU under different traces



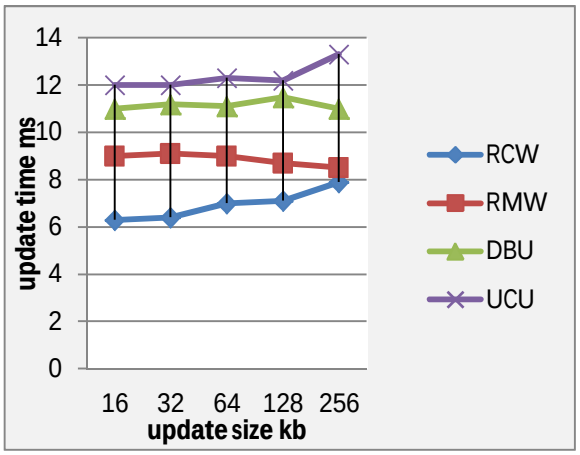
BLOCK SIZE(KB)

b).Update time for RCW,RMW,DBU UCU under different block size



TRACE NAME

c)Update time for RCW,RMW,DBU,UCU

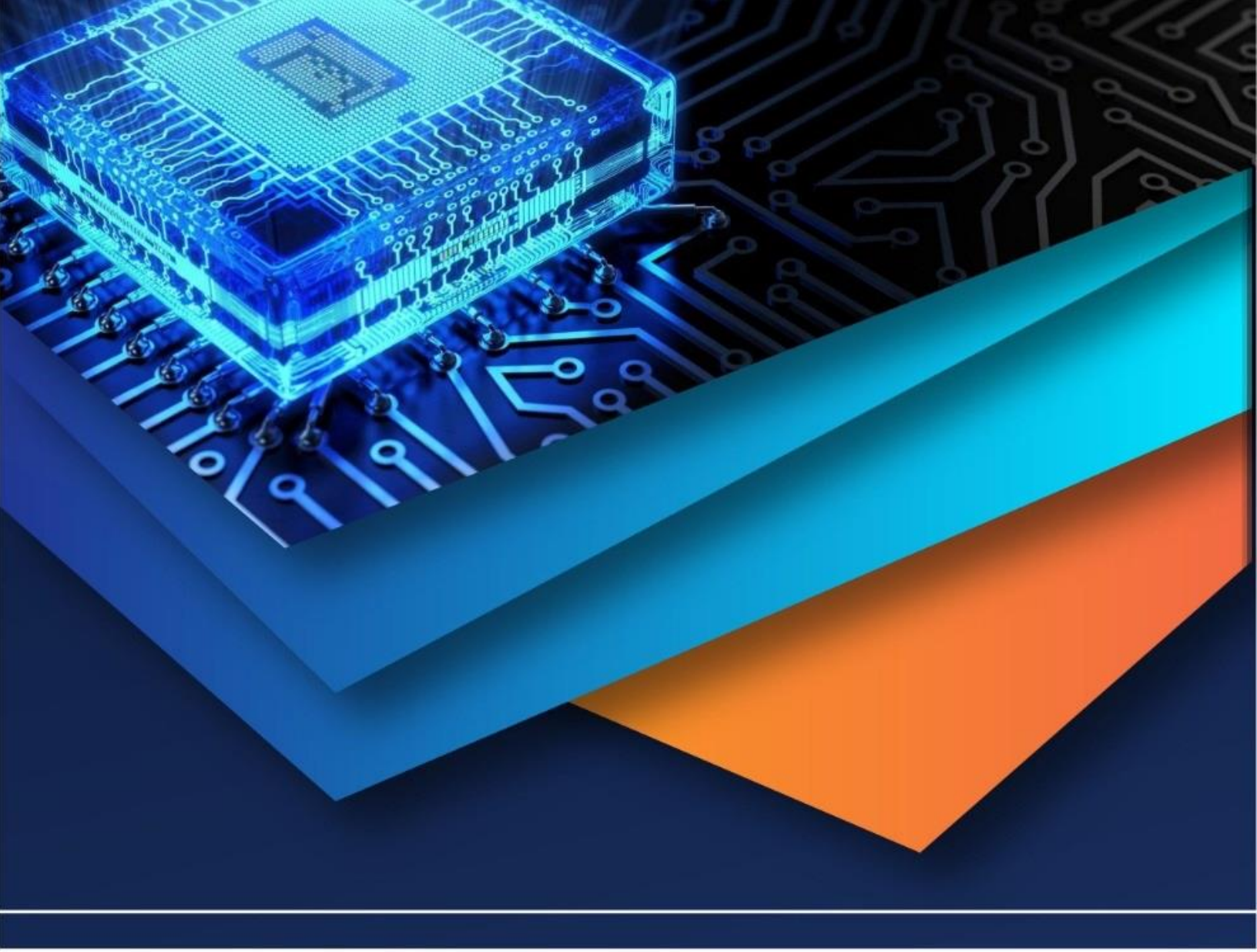


UPDATE SIZE KB

under different trace.

REFERENCES

[1] S. Reed and G. Solomon, "Polynomial codes over certain finite fields," J. Soc. Ind. Appl. Math., vol. 8, no. 2, pp. 300–304, 1960.
[2] J. S. Plank and L. Xu, "Optimizing Cauchy Reed-Solomon codes for fault tolerant network storage applications," in Proc. IEEE Int. Symp. Netw.Comp. Appl., 2006, pp. 173–180.
[3] M. Blaum, J. Brady, J. Bruck, and J. Menon, "EVENODD: An efficient scheme for tolerating double disk failures in RAID architectures," IEEE Trans. Comput., vol. 44, no. 2, pp. 192–202, Feb. 1995.
[4] P. Corbett, et al., "Row-diagonal parity for double disk failure correction," in Proc. 3rd USENIX Conf. File Storage Technol., 2004, pp. 1–14.
[5] Y. Fu and J. Shu, "D-Code: An efficient RAID-6 code to optimize I/O loads and read performance," in Proc. IEEE Int. Parallel Distrib. Process. Symp., 2015, pp. 603–612.
[6] Y. Wang, X. Yin, and X. Wang, "MDR codes: A new class of RAID-6 codes with optimal rebuilding and encoding," IEEE J. Sel. Areas Commun., vol. 32, no. 5, pp. 1008–1018, May 2014.
[7] J. S. Plank, "The RAID-6 Liberation codes," in Proc. 6th USENIX Conf. File Storage Technol., 2008, pp. 97–110.
[8] M. Blaum and R. M. Roth, "On lowest-density MDS codes," IEEE Trans. Inf. Theory, vol. 45, no. 1, pp. 46–59, Jan. 1997.
[9] Y. Zhu, J. Lin, P. P. C. Lee, and Y. Xu, "Boosting degraded reads in heterogeneous erasure-coded storage systems," IEEE Trans. Comput., vol. 64,no. 8, pp. 2145–2157, Aug. 2015.
[10] C. Huang and L. Xu, "STAR: An efficient coding scheme for correcting triple storage node failures," IEEE Trans. Comput., vol. 57, no. 7, pp. 889–901, Jul. 2007.
[11] Y. Fu, J. Shu, Z. Shen, and G. Zhang, "Reconsidering single disk failure recovery for erasure coded storage systems: Optimizing load balancing in stack-level," IEEE Trans. Parallel Distrib. Syst., vol. 27, no. 5, pp. 1457–1469,May 2015.
[12] C. Huang, et al., "Erasure coding in Windows Azure storage," in Proc.USENIX Annu. Tech. Conf., 2012, pp. 15–26.
[13] J. Shen, J. Gu, Y. Zhou, and X. Wang, "Cloud-of-clouds storage made efficient: A pipeline-based approach," in Proc. IEEE Int. Conf. Web Serv., 2016, pp. 724–727.
[14] J. C. Chan, Q. Ding, P. P. Lee, and H. H. Chan, "Parity logging with reserved space: Towards efficient updates and recovery in erasure-coded clustered storage," in Proc. 12th USENIX Conf. File Storage Technol., 2014,pp. 163–176.
[15] A. Thomasian, "Reconstruct versus read-modify writes in RAID," Inf. Process. Lett., vol. 93, no. 4, pp. 163–168, 2005.
[16] M.-L. Chiang and J.-S. Huang, "Improving the performance of log structured file systems with adaptive block rearrangement," in Proc. ACM Symp. Appl. Comput., 2007, pp. 1136–1140.
[17] Y. Zhang, C. Wu, J. Li, and M. Guo, "TIP-Code: A three independent parity code to tolerate triple disk failures with optimal update complexity," in Proc. 45th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw., 2015, pp. 136–147.
[18] Q. Chen, L. Liang, Y. Xia, and H. Chen, "Mitigating sync amplification for copy-on-write virtual disk," in Proc. 14th USENIX Conf. File Storage Technol., 2016, pp. 241–247.
[19] R. Verma, A. A. Mendez, S. Park, S. Mannar swamy, T. Kelly, and C. B. Morrey III, "Failure-atomic updates of application data in a Linux file system," in Proc. 13th USENIX Conf. File Storage Technol., 2015, pp. 203–211.
[20] D. Le, H. Huang, and H. Wang, "Understanding performance implications of nested file systems in a virtualized environment," in Proc. 10th USENIX Conf. File Storage Technol., 2012, pp. 1–13.
[21] P. H. Carns, R. B. Ross, and R. Thakur, "PVFS: A parallel file system for Linux clusters," in Proc. 4th Annu. Linux Showcase Conf., 2000, pp. 391–430.
[22] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in Proc. 7th USENIX Symp. Operating Syst. Design Implementation, 2006, pp. 307–320.
[23] M. N. D. Santos and R. Cerqueira, "GridFS: Targeting data sharing in grid environments," in Proc. 6th IEEE Int. Symp. Cluster Comput. Grid, 2006, Art. no. 17.
[24] J. L. Hafner, V. Deenadhayalan, K. Rao, and J. A. Tomlin, "Matrix methods for lost data reconstruction in erasure codes," in Proc. 4th USENIX Conf. File Storage Technol., 2005, pp. 183–196.
[25] J. S. Plank, C. D. Schuman, and B. D. Robison, "Heuristics for optimizing matrix-based erasure codes for fault-tolerant storage systems," in Proc.42nd Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw., 2012, pp. 1–12.
[26] C. Huang, J. Li, and M. Chen, "On optimizing XOR-based codes for fault-tolerant storage applications," in Proc. Inf. Theory Workshop, 2007, pp. 218–223.
[27] Y. Zhu, P. P. C. Lee, Y. Xu, Y. Hu, and L. Xiang, "On the speedup of recovery in large-scale erasure-coded storage systems," IEEE Trans. Parallel Distrib. Syst., vol. 25, no. 7, pp. 1830–1840, Jul. 2014.
[28] L. Xiang, Y. Xu, J. Lui, and Q. Chang, "Optimal recovery of single disk failure in RDP code storage systems," in Proc. ACM SIGMETRICS Perform. Eval. Rev., vol. 38, no. 1, 2010, pp. 119–130.
[29] O. Khan, R. Burns, J. S. Plank, W. Pierce, and C. Huang, "Rethinking erasure codes for cloud file systems: Minimizing I/O for recovery and degraded reads," in Proc. 10th USENIX Conf. File Storage Technol., 2012,pp. 1–14.
[30] N. B. Shah, K. V. Rashmi, P. V. Kumar, andK. Ram chandran, "Distributed storage codes with repair-by-transfer and non achievability of interior points on the storage-bandwidth tradeoff," IEEE Trans. Inf. Theory, vol. 58, no. 3, pp. 1837–1852, 2012.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY