International Journal of Multidisciplinary Research And Studies

Available online at www.ijmras.com

INTERNATIONAL JOURNAL OF MULTIDISCIPLINARY RESEARCH AND STUDIES
ISSN: 2640 7272
Volume:07; Issue:02 (2024)
Doi: 10.33826/ijmras/v07i02.4

# Utilizing real service simulation techniques to investigate distributed storage systems' latency performance based on an examination of Amazon S3

SU RIGUGE[1], DR MOHAMMAD NIZAMUDDIN INAMDAR [2a]
PhD Research Scholar in Engineering, Lincoln University College Malaysia
Professor in Lincoln University College Malaysia
Contact Details: [a] nizamuddin@lincoln.edu.my

## Abstract

Current erasure codes rely heavily on data nodes to generate the parity nodes. The greater the tolerance for error, and the more "If we can increase the number of parity nodes, we may increase our chances of restoring the original data. As the number of parity nodes grows, the storage overhead will rise, and the repair burden on data nodes will rise as well, because data nodes are queried often to help in the repair of parity nodes. If a global parity node fails in LRC [25, 26], for instance, all data nodes must be fixed. It will take more time to process read requests for data nodes as a result of the "increasing demands on the network's data nodes. An application where frequent data retrievals are unwelcome is a Google search.
In an effort to cut down on waiting time, "produces both data and parity nodes, the latter of which can take over part of the repair work normally done by the former. In other words, the number of data nodes that may be accessed remains constant, regardless of whether or not a parity node is functioning. It would appear that parity nodes incur additional storage costs. Generating parity nodes using parity nodes can assist decrease access latency without raising or decreasing the storage requirements if the architecture is sound ", as we shall show in the following sections [27, 28], above your head.

In this research, we'll compare and contrast the effectiveness of "Hierarchical Tree Structure Code (HTSC) and High Failure-tolerant Hierarchical Tree Structure Code (FH HTSC)."

**Keyword:** Hierarchical Tree Structure, Data Retrieval, Data Nodes.

## INTRODUCTION

Searching, social networking, and e-commerce have all seen significant increases in popularity over the previous decade. Every day, we produce a massive amount of digital data. Research and business alike are grappling with the difficulty of designing cost-effective storage systems. The data explosion has necessitated the creation of large-scale distributed storage systems. The Hadoop Distributed File System (HDFS) [2] and Windows Azure Storage (WAS) [3] are just a few examples. Using these storage systems, huge data, high speed computing, and cloud-scale applications may be met with great reliability and ubiquity. It is common for a large distributed storage system to be developed utilising a large number of inexpensive and unstable storage devices, and these individual nodes are susceptible to failures. There are substantial advantages to these systems in terms of scalability, yet failure is the rule, not the exception. [1] As a result, we must overcome frequent system failures and guarantee that these systems are reliable and resilient.

## LITERATURE REVIEW

In large-scale distributed storage systems, redundancy is provided using replication or erasure coding, which provides a high level of failure protection.

GFS ensures that data may be accessed reliably by distributing the information among three separate storage nodes. Google's frequent read requirements can easily be accommodated by this simple replication approach [6]. Because of the large storage requirements for a given degree of fault tolerance, replication maintains data availability and combats data losses in the event of node failures.

Files of fixed size M can be divided into k parts (sometimes referred to as "k nodes"), each of size M, and encoded into n encoded nodes for use in generic erasure code systems. In comparison to replication, the storage requirements for a given degree of dependability can be significantly reduced using the erasure coding approach. Reed Solomon (RS) codes, for example, are among the most widely used and most efficient storage codes because of their Maximum-Distance-Separable (MDS) characteristic [4].

It is an element of a standard code that has a codeword. There are n nodes in an MDS codeword, and any k of those nodes can be used to reassemble the entire text. In addition, we refer to a codeword as a systematic code if it comprises original data nodes. There are k original data nodes and an equal number of n-k parity nodes in any feasible MDS codeword [5]. Nodes of a codeword are typically kept on multiple storage devices in different locations to avoid failures due to common reasons.

Any three of the six nodes in a (6,3) MDS codeword may decode all of the information in the codeword, as depicted in Fig. 1.1. Due to the uncoded nature of d1 through d3, the code is logical. Large-scale distributed storage systems using coding often use a code with a predetermined set of (n, k) parameters and a specified size for each codeword to store its data, making it easier to maintain and operate. There are two types of RS codes that are utilised in HDFS and GFS II: (14,10) for Face-HDFS book's and (9,6) for GFS II [7, 8]. For actual large-scale distributed storage systems, this means a codeword comprises several files of fixed total size. We can better investigate the storage system's characteristics because of the constant coding rate.

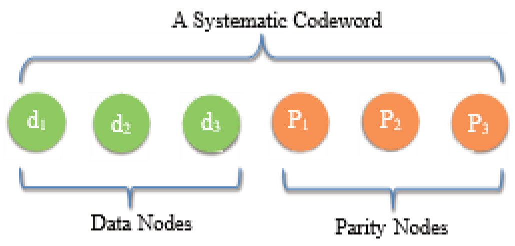

Figure 1.1 A codeword of systematic (6,3) MDS code.

## STATEMENT OF THE PROBLEM

User experience may be considerably affected by the latency of access in distributed storage systems. Erase codes have long been known to be more reliable than replication at the same storage cost, but it has only recently been shown that coding may minimise access latency [9]. Chinese researchers proved the first time that codes can minimise queueing delay [10]. After that, a great deal of study has focused on the Redundant Scheme (RedS), which indicates that sending redundant requests to storage systems may minimise latency. While part of the work [11] is theoretical, others [12] use trace-driven simulations to test the findings. To learn more about how duplicate queries can assist minimise latency, see Shah et al. [13]. In [14, 15], RedS and RanS latency-cost tradeoffs will be examined. Coding surpasses replication in terms of delay under the same storage cost, according to [16].

Despite these attempts, while evaluating latency performance, certain important practical requirements of distributed storage systems are overlooked.

When users request files of varying sizes from a codeword, [17, 18] can only handle cases where each request reads the whole content of the codeword. No comparison will be made between replication and coding when a request to the replication system may want files from more than one data node at the same time, as will be done in the case of single node reads in [19]. A more generalised version of the second situation is more in line with current large-scale storage systems.

Most earlier studies [20, 21, 22] assume pure ex-ponential service times, but our real-world observations on Amazon S3 show that this assumption falls well short of the real-world reality, as demonstrated by Liang et al.

Repairing failures is a common task in distributed storage systems, in addition to the typical data retrieval [23]. GFS, Amazon S3, and WAS are three examples of distributed storage systems that rely on a large number of faulty hardware. Recovery procedures require roughly 180TB per day of data transfer between racks in Facebook HDFS, and there are multiple times of high repair rate each day [24]. Increased frequency of read requests will certainly lead to an increase in access latency under the same conditions since there are fewer nodes to store the data requested by a read request. No data is lost when a storage node fails and repair requests are given higher priority than read/write operations to ensure that no data is lost. Because repair requests are prioritised above read requests, the access times for such requests may be significantly impacted. As a result, in this study, we will investigate the influence of repair requests on read request access latency.

### Objective of the Study

- To find the best methods for direct readings in order to minimise latency.

### Research Questions

- Which one is the best methods for direct readings in order to minimise latency?

## RESEARCH METHODOLOGY

Using erasure codes and replications in distributed storage systems is one way to deal with system failures [29]. Codes often used in practise are generally systematic codes, which implies each codeword contains a duplicate of the original data. It is also possible to use erasure coding in Windows Azure storage (WAS) systems, but only when a file reaches a specific size (e.g., 3GB). If you're looking for just a portion of the file, the storage nodes will be able to get it from one of the codeword's huge files, which are often extremely large in practise (we call those requests direct reads) [31, 32, 33]. Requests for k-access reads, in which each request must read the whole file in a codeword and must access at least k nodes, are another type of requests. A distributed storage system's latencies will vary depending on the number of direct and k-access reads that are performed. To our knowledge, this is the first time that direct readings have been explored in detail in any previous research [30].

Latency performance is crucial in distributed storage systems, and some studies claim that codes can minimise latency in data centres, while many other strategies have been proposed to reduce latency in distributed storage systems.. Previous research has mostly ignored direct readings and only looked at k-access accesses. There hasn't been any research done on how RedS can speed up direct readings. While RedS sends requests to all n nodes for each k-access read, the Random Scheme (RanS) only randomly sends requests to those k nodes. Compared to RanS, RedS requires a larger investment of time and resources. When it comes to practical distributed storage systems, RanS is a popular choice since it is easy to implement and does not require additional information or resources.

## RESEARCH DESIGN

Redundant Request Technique (RedS) is a lately popular read scheme for (n, k) MDS-coded storage systems [34, 35, 36]. No matter how many files a read request requests in a codeword, RedS splits it into n jobs and sends them to each of the n nodes. When k nodes out of n have finished providing their services, the request is considered complete, and the remaining n k jobs are promptly terminated.

We've developed a solution based on RedS that can manage requests for files of varying sizes while also reducing access latency. Flexible Redundant Scheme is what we call it (FRedS).

## DATA ANALYSIS

Generally speaking, "In order to save your data using HTSC(D) or FH HTSC, you will need to combine your files into a single large one of size M, say 1 to 3 GB, and then divide it into K parts (D, h). The fixed size M may be calculated using the available storage space at each node and the parameters of HTSC(D, h) or FH FH HTSC. (D, h). Users are often only interested in a subset of a "file's uncoded systematic component, which is kept in one of the K nodes. Earlier studies assumed that readers would desire access to the entire contents of a "Considering that every bit of information is now stored in the K-tree, the use of a codeword to describe it is a major shift. However, this oversimplifies things rather than reflecting reality. In WAS, for instance, erasure coding is available only for files beyond a certain size threshold (say, 3GB) [31]. Most people only need a small percentage of the 3GB available, so it's not surprising that it's a waste. This conforms to the HTSC's planned functionality (D, h). For this reason, we will be concentrating on read requests from customers that are only interested in a subset of the information stored in one of the K data nodes "inferences drawn from this research.

## CONCLUSION

Generally speaking, "In order to save your data using HTSC(D) or FH HTSC, you will need to combine your files into a single large one of size M, say 1 to 3 GB, and then divide it into K parts (D, h). The fixed size M may be calculated using the available storage space at each node and the parameters of HTSC(D, h) or FH FH HTSC. (D, h). Users are often only interested in a subset of a "file's uncoded systematic component, which is kept in one of the K nodes. Earlier studies assumed that readers would desire access to the entire contents of a "Considering that every bit of information is now stored in the K-tree, the use of a codeword to describe it is a major shift. However, this oversimplifies things rather than reflecting reality. In WAS, for instance, erasure coding is available only for files beyond a certain size threshold (say, 3GB) [31]. Most people only need a small percentage of the 3GB available, so it's not surprising that it's a waste. This conforms to the HTSC's planned functionality (D, h). For this reason, we will be concentrating on read requests from customers that are only interested in a subset of the information stored in one of the K data nodes "inferences drawn from this research.

## LIMITATIONS OF THE STUDY

As the nodes and connections must be protected, it is challenging to guarantee effective security in distributed systems. Data and messages might be lost in the network as they travel between nodes. In comparison to a single user system, the database related to distributed systems is highly complex and difficult to handle. In the event that all of the distributed system's nodes attempt to communicate data at once, the network may get overloaded.

## REFERENCES

[1]. S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," in ACM SIGOPS Operating Systems Review, vol. 37, no. 5. ACM, 2003, pp. 29–43.
[2]. M. Foley, "High availability HDFS," in 28th IEEE Conference on Massive Data Storage, MSST, vol. 12, 2012.
[3]. C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li,S. Yekhanin et al., "Erasure coding in Windows Azure storage," in USENIX ATC, 2012, pp. 15–26.
[4]. N. B. Shah, K. Lee, and K. Ramchandran, "The MDS queue: Analysing the latency performance of erasure codes," in IEEE International Symposium on Information Theory (ISIT), 2014, pp. 861–865.

[5]. B. Y. Kong, J. Jo, H. Jeong, M. Hwang, S. Cha, B. Kim, and I.-C. Park, “Low- complexity low-latency architecture for matching of data encoded with hard systematic error-correcting codes,” IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 22, no. 7, pp. 1648–1652, 2014.

[6]. K. Rashmi, N. B. Shah, D. Gu, H. Kuang, D. Borthakur, and K. Ramchan- dran, “A solution to the network challenges of data recovery in erasure-coded distributed storage systems: A study on the Facebook warehouse cluster,” in Presented as part of the 5th USENIX Workshop on Hot Topics in Storage and File Systems. USENIX, 2013.

[7]. A. Fikes, “Storage architecture and challenges,” Talk at the Faculty Summit, 2010.

[8]. D. Ford, F. Labelle, F. I. Popovici, M. Stokely, V.-A. Truong, L. Barroso,C. Grimes, and S. Quinlan, “Availability in globally distributed storage sys- tems.” in OSDI, 2010, pp. 61–74.

[9]. A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchan- dran, “Network coding for distributed storage systems,” IEEE Transactions on Information Theory, vol. 56, no. 9, pp. 4539–4551, 2010.

[10]. K. V. Rashmi, N. B. Shah, and P. V. Kumar, “Optimal exact-regenerating codes for distributed storage at the msr and mbr points via a product-matrix construction,” IEEE Transactions on Information Theory, vol. 57, no. 8, pp. 5227–5239, 2011.

[11]. V. R. Cadambe, S. A. Jafar, H. Maleki, K. Ramchandran, and C. Suh, “Asymp- totic interference alignment for optimal repair of MDS codes in distributed data storage,” 2011.

[12]. N. B. Shah, K. Rashmi, P. V. Kumar, and K. Ramchandran, “Explicit codes minimizing repair bandwidth for distributed storage,” in Information Theory Workshop (ITW), 2010 IEEE. IEEE, 2010, pp. 1–5.

[13]. V. R. Cadambe, S. A. Jafar, and H. Maleki, “Distributed data storage with minimum storage regenerating codes-exact and functional repair are asymp- totically equally efficient,” arXiv preprint arXiv:1004.4299, 2010.

[14]. N. B. Shah, K. Rashmi, P. V. Kumar, and K. Ramchandran, “Interference alignment in regenerating codes for distributed storage: Necessity and code constructions,” IEEE Transactions on Information Theory, vol. 58, no. 4, pp. 2134–2158, 2012.

[15]. A. Duminuco and E. Biersack, “A practical study of regenerating codes for peer-to-peer backup systems,” in 29th IEEE International Conference on Dis- tributed Computing Systems. IEEE, 2009, pp. 376–384.

[16]. A. Duminuco and E. W. Biersack, “Hierarchical codes: A flexible trade-off for erasure codes in peer-to-peer storage systems,” Peer-to-peer Networking and Applications, vol. 3, no. 1, pp. 52–66, 2010.

[17]. M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali,S. Chen, and D. Borthakur, “Xoring elephants: Novel erasure codes for big data,” in Proceedings of the 39th international conference on Very Large Data Bases. VLDB Endowment, 2013, pp. 325–336.

[18]. J. Li and B. Li, “Erasure coding for cloud storage systems: A survey,” Ts- inghua Science and Technology, vol. 18, no. 3, pp. 259–272, 2013.

[19]. A. G. Dimakis, K. Ramchandran, Y. Wu, and C. Suh, “A survey on network codes for distributed storage,” Proceedings of the IEEE, vol. 99, no. 3, pp. 476–489, 2011.

[20]. A. Rudra, P. K. Dubey, C. S. Jutla, V. Kumar, J. R. Rao, and P. Rohatgi, "Ef- ficientRijndael encryption implementation with composite field arithmetic," in Cryptographic Hardware and Embedded Systems?CHES 2001. Springer, 2001, pp. 171–184.

[21]. J. Brutlag, "Speed matters for Google web search," Google. June, 2009.

[22]. L. Huang, S. Pawar, H. Zhang, and K. Ramchandran, "Codes can reduce queueing delay in data centers," in IEEE International Symposium on Infor- mation Theory (ISIT), 2012, pp. 2766–2770.

[23]. N. B. Shah, K. Lee, and K. Ramchandran, "When do redundant requests re- duce latency?" in the 51st Annual Allerton Conference on Communication, Control, and Computing. IEEE, 2013, pp. 731–738.

[24]. G. Liang and U. C. Kozat, "TOFEC: Achieving optimal throughput-delay trade-off of cloud storage using erasure codes," in Proceedings of INFOCOM. IEEE, 2014, pp. 826–834.

[25]. G. Joshi, Y. Liu, and E. Soljanin, "On the delay-storage trade-off in content download from coded distributed storage systems," IEEE Journal on Selected Areas in Communications, vol. 32, no. 5, pp. 989–997, 2014.

[26]. Y. Xiang, T. Lan, V. Aggarwal, and Y. F. R. Chen, "Joint latency and cost optimization for erasurecoded data center storage," ACM SIGMETRICS Per- formance Evaluation Review, vol. 42, no. 2, pp. 3–14, 2014.

[27]. B. Li, A. Ramamoorthy, and R. Srikant, "Mean-field-analysis of coding versus replication in cloud storage systems," in Proceedings of INFOCOM. IEEE, 2016.

[28]. G. Liang and U. C. Kozat, "Fast Cloud: Pushing the envelope on delay perfor- mance of cloud storage with coding," IEEE/ACM Transactions on Network- ing, vol. 22, no. 6, pp. 2012–2025, 2014.

[29]. G. Ananthanarayanan, S. Agarwal, S. Kandula, A. Greenberg, I. Stoica,D. Harlan, and E. Harris, "Scarlett: coping with skewed content popularity in mapreduce clusters," in Proceedings of the sixth conference on Computer systems. ACM, 2011, pp. 287–300.

[30]. A. Kala Karun and K. Chitharanjan, "A review on hadoophdfs infrastructure extensions," in Conference on Information & Communication Technologies (ICT). IEEE, 2013, pp. 132–137.

[31]. M. Harchol-Balter, Performance Modeling and Design of Computer Systems: Queueing Theory in Action. Cambridge University Press, 2013.

[32]. H. A. David and H. N. Nagaraja, Order statistics. Wiley Online Library, 1981.

[33]. M. Rahman and L. Pearson, "Moments for order statistics in shift parameter exponential distribution," Journal of Statistical Research, vol. 36, no. 1, pp. 75–83, 2002.

[34]. S. B. Wicker and V. K. Bhargava, Reed-Solomon codes and their applications. John Wiley & Sons, 1999.

[35]. Q. Shuai, V. O. K. Li, and Y. Zhu, "Performance models of access latency in cloud storage systems," in Fourth Workshop on Architectures and Systems for Big Data, 2014.

[36]. M. Blaum, J. Brady, J. Bruck, and J. Menon, "Evenodd: An efficient scheme for tolerating double disk failures in raid architectures," IEEE Transactions on Computers, vol. 44, no. 2, pp. 192–202, 1995.

[37]. L. Xu and J. Bruck, “X-code: Mds array codes with optimal encoding,” IEEE Transactions on Information Theory, vol. 45, no. 1, pp. 272–276, 1999.

[38]. P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, andS. Sankar, “Row-diagonal parity for double disk failure correction,” in Pro- ceedings of the 3rd USENIX Conference on File and Storage Technologies, 2004, pp. 1–14.