An Analysis Based on Amazon S3 That Makes Use of Real-World Service Simulation Techniques Is Presented in This Study, Which Aims to Investigate the Latency Performance of Distributed Storage Systems

Suriguge[1*], Noraisyah Binti Tajudin[2]

1*Research Scholar Lincoln University College Malaysia
2Lincoln University College Malaysia
Email: Suriguge@Lincoln.Edu.My

Abstract

The generation of parity nodes is strongly dependent on data nodes in the erasure codes that are currently in use. The higher the tolerance for mistake, and the more people are willing to It is possible that our chances of successfully recovering the original data will improve if we are able to increase the number of parity nodes as well. The storage overhead will increase as the number of parity nodes increases, and the repair load on data nodes will also increase. This is due to the fact that data nodes are queried often in order to assist in the repair of parity nodes at the same time. In the event that a global parity node fails in LRC [25, 26], for example, it is necessary to solve all of the data nodes. As a consequence of the "increasing demands on the network's data nodes," the amount of time required to process read requests for data nodes would increase more than before. Google search is an example of an application that should not be used for retrieving data on a regular basis. "Produces both data and parity nodes, it is possible for the latter to take over some of the repair work that is normally done by the former. This is done in an effort to reduce the amount of time that is spent waiting." To put it another way, the number of data nodes that may be accessed does not change under any circumstances, regardless of whether or not a parity node is operational. When it comes to storage costs, it would seem that parity nodes suffer extra expenses. If the design is correct, generating parity nodes by employing parity nodes may help reduce access latency without increasing or lowering the storage needs. This is something that we will demonstrate in the coming sections, which are over your head.

Keyword: *Hierarchical Tree structure, data retrieval, data nodes.*

1. Introduction

The last ten years have seen a substantial rise in the popularity of a number of online activities, including searching, social networking, and online shopping. Over the course of each day, we generate an enormous quantity of digital data. Both the commercial world and the research world are struggling with the challenge of building storage systems that are both cost-effective and efficient (M. Foley, 20128). Because of the increase in the amount of data, it has become necessary to develop large-scale distributed storage systems. To name just a few instances, there is the Hadoop Distributed File System (HDFS) and Windows Azure Storage (WAS]. With the help of these storage systems, it is possible to meet the requirements of cloud-scale applications, high-speed computation, and massive amounts of data with excellent dependability and ubiquity. When developing a large, distributed storage system, it is usual practice to make use of a large number of storage devices that are both affordable and unstable. These individual nodes are subject to failures, and it is typical for storage devices to be unstable. Failure is the norm rather than the exception

when it comes to these systems, despite the fact that they provide significant benefits in terms of expanding their capacity (K. Rashmi, 2019) As a consequence of this, we face the challenge of overcoming frequent system failures and ensuring that these systems are both dependable and robust.

2. Background of the Study

Redundancy, in the form of replication or erasure coding, offers a high degree of failure protection in large-scale distributed storage systems (P. Gopalan,2019).

GFS distributes the information across three distinct storage nodes, guaranteeing that it may be retrieved reliably. It's easy to meet Google's frequent reading criteria using this straightforward replication strategy. Replication" keeps data available and prevents data loss when nodes fail due to the high storage needs for a certain level of fault tolerance.

To implement general erasure coding systems, files with a given size M may "be partitioned into k sections (also referred to as "k nodes"), each with a size of M, and encoded into n encoded nodes. For a given level of reliability, the storage needs may be drastically decreased utilising the erasure" coding strategy compared to replication. For instance, because to their Maximum-Distance-Separable (MDS) property, Reed Solomon (RS) codes are among the most popular and effective storage codes. “Standard code that has a codeword" describes this component. Each MDS codeword has n nodes, and any combination of those k nodes may be used to reconstruct the whole text (G. Joshi, 2019). Furthermore, when a codeword contains original data nodes, we call it a systematic code. In every possible MDS codeword, there are k original data nodes and an equal

number of "n-k parity" nodes. It is standard practice to store codeword nodes on separate storage devices in various places to reduce the likelihood of failures caused by "common" causes. Shows that any three of the six nodes that make up a “MDS codeword” may decode the whole codeword.1. The code makes sense since d1 through d3 are not coded. Coding-based large-scale distributed storage systems typically use a code with a fixed size for each codeword and a predefined set of (n, k) parameters to facilitate operation and maintenance. For Face-HDFS books, HDFS uses RS codes (14.0) and for GFS II, it uses RS codes (9.6). A codeword consists of many files with a set total size in real-world large-scale distributed storage systems. The consistent coding rate allows us to "better investigate" the features of the storage system.

3. purpose of the research

Current erasure codes mostly use data nodes to generate the parity nodes. It is possible to increase failure tolerance and the number of alternatives to recover the original data by increasing the number of "parity nodes" using this strategy (R. Nelson, 2019. Increasing the number of parity nodes will lead to higher storage overhead and a heavier repair burden on data nodes since they are often used to aid with parity node repairs. For instance, in LRC, if a global parity node fails, all data nodes must be repaired. The "increased workload on data nodes" causes read requests to take more time to process. A programme that often retrieves data is not always desirable, such as.

One way to cut down on wait times is to generate parity nodes alongside data nodes, which will transfer part of the repair work from data nodes to them. That is, we may increase the number of data nodes that can be accessed by simply replacing a failed parity node with another parity node. However, it seems that the storage overhead for parity nodes is larger. As we shall see in the next sections, it is possible to generate parity nodes with parity nodes to reduce storage overhead without raising or decreasing access latency with a good design (G. Liang,2018).

Researchers in this research will look at how well "Hierarchical Tree Structure Code (HTSC) and High Failure-tolerant Hierarchical Tree Structure Code (FH HTSC) will" work.

4. Literature Review

Redundancy is given in large-scale distributed storage systems by the use of replication or erasure coding, which offers a high degree of protection against failure (S. Chen,2018).

Through the distribution of the information among three distinct storage nodes, the GFS system guarantees that data may be retrieved in a dependable manner. The frequent read needs that Google has may be readily handled by using this simple replication strategy [6]. The use of replication helps to ensure that data is always accessible and prevents data loss in the event that a node fails. This is because replication requires a significant amount of storage space for a relatively high level of fault tolerance.

"Files of fixed size M can be divided into k parts (sometimes referred to as "k nodes"), each of which is of size M, and encoded into n encoded nodes for use in generic erasure code systems," according to the definition of the term. Through the utilisation of the erasure" coding strategy, it is possible to drastically lessen the amount of storage that is necessary for a given level of dependability in contrast to the replication method. As an instance, Reed Solomon (RS) codes are among the most extensively used and most effective storage codes due to the fact that they possess the Maximum-Distance-Separable (MDS) property (Q. Shuai,2017).

A codeword is a component of a "standard code," which is an element of the code. In an MDS codeword, there are n nodes, and any k of those nodes may be used to reassemble the full text regardless of which node is chosen. A further point to consider is that a codeword is considered to be a systematic code if it contains the original data nodes. Every viable MDS codeword has k original data nodes plus an equal number of "n-k parity" nodes [5]. This is the case regardless of the codeword. In order to prevent failures that are caused by common circumstances, it is standard practice to store the nodes of a codeword on various storage devices located in separate places.

According to the diagram in Figure 1.1, any three of the six nodes that make up a (6,3) "MDS codeword” have the ability to decode all of the information contained inside the codeword. In light of the fact that d1 through d3 are not coded, the code is rational. When it comes to storing its data, large-scale distributed storage systems that make use of coding often use a code that has a preset set of (n, k) parameters and a fixed size for each codeword. This makes the system simpler to manage and run. In HDFS and GFS II, there are two different kinds of RS codes that are used: (14,10) for Face-HDFS books and (9,6) for GFS II. Both of these codes are used to store data. According to this definition, a codeword is comprised of several files that have a set total size when applied to genuine large-scale distributed storage systems. Because of the consistent coding rate, we are able to conduct a more thorough investigation of the properties of the storage system.

5. Research Questions

1. What "are the characteristics of latency in direct" readings?
2. Which one is the "best methods for direct readings in order to minimise" latency?

3. Is there "any correlation between latency performance of direct" reads and degraded reads.

6. Methodology

When dealing with system failures, one method that may be used in distributed storage systems is the utilisation of erasure codes and replications. Generally speaking, codes that are often used in practice are systematic codes, which means that each codeword comprises a copy of the data that was originally collected. There is also the possibility of using erasure coding in Windows Azure storage (WAS) systems; however, this is only allowed when a file exceeds a certain size, such as three gigabytes. In the event that you are just interested in a specific section of the file, the storage nodes will be able to get it from one of the enormous files that the codeword operates with. These files are often quite big in practice (we refer to these requests as direct reads). The requests for k-access reads are another form of requests. In this sort of request, each request must read the whole file in a codeword and must access at least k nodes. The amount of direct and k-access reads that are carried out via a distributed storage system will determine the latencies that are experienced by the system. To the best of our knowledge, this is the very first time that direct readings have been investigated from a comprehensive standpoint in any of the prior studies.

Latency performance is considered "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" system configurations. Previously conducted research has, for the most part, disregarded direct readings and has solely focused on k-access information. To our knowledge, there has been no investigation into the ways in which RedS may expedite direct readings.The Random Scheme (RanS) only sends requests to those k nodes in a random fashion, in contrast to RedS, which sends requests to all n nodes for each k-access read. When compared to RanS, RedS necessitates a greater time and resource commitment that must be made. When it comes to realistic distributed storage systems, RanS is a popular choice since it is simple to deploy and does not need any extra information or resources. This feature makes it an attractive option.

7. Results

To simplify matters, we assume a homogeneous situation in which the direct read arrival rate for the content of each data node is the same and the proportion of direct read task for each data node is likewise the same, namely x. General read arrival rate for data and parity nodes may be obtained with little effort using the formulas iJ = x + (1 x)(k 1)p, where i = 1, 2,..., k, and p is the parity bit.

If we write j = k + 1, k + 2, n we get jJ = (1 x) kp.

We analyse the practical issue that there is no direct read to parity nodes, and while the results are comparable to our prior work we make a clear distinction between the general read arrival rate of data and parity nodes.Latency for direct readings can be reduced by performing degraded read jobs in a systematic (n, k) MDS-coded storage system under the homogeneous condition if and only if the code rate is n > k 1 + 1 and k x 1. However, under these conditions, the latency for direct reads cannot be reduced if the code rate is k k n+k1 k 1 + and 0Proof. If all other factors remain constant, the latency in a distributed storage system will increase as the average rate at which reads arrive increases. Comparison is made between the performance of degraded readings and that of direct read jobs alone (x = 1) in terms of reducing delay. If x = 1, then the delay solely" "depends on the data nodes, as the general read ar- rival rate for each data node is and that for each parity one is 0. Reduced read performance occurs in the range 0–1. When p(k1) 1, the average read arrival rate at a data node iJ, as suggested by Eq we are able to delay feedback for each data node. In light of p = k, we may write the condition as code rate n1 n > k1 + 1. One may alternatively get iJ = 1 + (1 x) p + (1 x) kip from Eq.

By plugging these values into Eq. (3.6), we obtain the connection between the average data read arrival rate and parity nodes, which is given by iJ = x (1 x) p + jJ. As long as x (1 x) p 0, the average read arrival rate of parity nodes is less than or equal to that of data nodes, meaning that the two types of nodes may achieve at most the same delay. By changing p to k, we get k x 1, which we can use to derive the condition for n1 as well.Lower latency is achieved by the data nodes already when n > k 1 + 1.Reduced latency is achieved by all nodes performing degraded read operations relative to the case where x = 1. Direct read latency can be reduced by performing degraded read operations when the coding rate is n > k 1 + 1 and k x 1.

When k = 1 and x = 0, we can similarly show that degraded read tasks will not result in a decrease in latency when k = n+k1.Degraded read activities can undoubtedly save latency, but we can't just switch over as many direct readings as feasible to them because doing so might incur significantly greater bandwidth costs. It is not possible to ensure that degraded read jobs will be successful in reducing latency when the code rate is n > k 1 + 1 and 0 x k, or when the code rate is k k n+k1 k1+ and x 1. These results are consistent with the results of Theorem 3.1. Given these unknowables, kkn+k1, the efficiency of degraded read jobs in lowering latency is not guaranteed. For this reason, a realistic method is required that may swiftly decrease latency by performing degraded read jobs.

Throughput is increased, latency is decreased, and server overload is prevented when load balancing is used. The primary concept is to offload work

from overworked servers onto less busy ones. Researchers have spent a lot of time looking at ways to implement replication-based load balancing in large-scale distributed storage systems like the Google File System (GFS) the Hadoop Distributed File System (HDFS) and others. In addition, load balancing is crucial for k-access readings since it aids in maximising the utilisation of all the nodes in a codeword. In order to accomplish load balance for direct readings, some of them must be transferred to degraded read jobs. Based on what has been discussed so far in this study, it appears that degraded read workloads will not help with latency reduction for direct readings but will instead raise the bandwidth cost. As a result, utilising load balancing to lessen wait times for direct readings necessitates a cost-effective and efficient technique.In practical distributed storage systems within a codeword, the probability of exactly one hot data node is much higher than that of more than one. In fact, if there is more than one hot data node in a codeword, the workload within that codeword is inevitably intensely heavy, resulting in extremely high latency which is probably unacceptable to users. In such cases, we usually quickly adjust the storage system to ensure that there is at most one hot data node in a codeword. Accordingly, in this work, we focus on the most likely case that there is at most one hot data node in a codeword.

In a codeword without hot data node, each data node has a small direct read arrival" rate.

"It's possible that degraded read operations can help cut down on latency by taking use of the spare resources in a codeword by shifting some of the load to parity nodes. To simplify matters, let's assume that the value of xi, where i = 1, 2,..., k, is the same, say x, for all data nodes that are relatively near to 1.When one data node in a codeword becomes hot, we aim to lower its general read arrival rate by reducing its probability to join the degraded reads of other data nodes and decreasing its x to transfer more of its direct reads to other nodes via degraded reads. Without loss of generality, suppose the hot data node is the first node in the codeword. Let λ1 and x1 denote its direct read arrival rate and fraction of direct read tasks, respectively. Suppose the average direct read arrival rate of the

other k − 1 data nodes is λo. With Eq. (3.1), we can easily get the general read

arrival rate of the hot data node as

λJ1 = x1λ1 + (1 − x) (k − 1) p1λo, (3.7) where p1 is the probability of the hot data node joining the degraded reads of other data nodes and p1 = k in (n, k) MDS coded storage systems.

We can also get the average general read arrival rate of the other k − 1 data nodes

o

λoJ =xλo + (1 − x) (k − 2) poλo + (1 − x1) pλ1

— (1 − x) k − 1 p λ,

(3.8)

n − 1

where p is the probability of the other n − 1 nodes joining the degraded reads of the

hot data node and p = k.

n−1We can obtain Eq. (3.8) as follows: on the right-hand side of Eq. (3.8), the sum of the first three terms represents the average general read arrival rate of the other

k − 1 data nodes when the hot data node does not join any degraded read of the

a non-hot data node from the set of k data nodes. So, po = k makes sense. If the hot data node" "is added to the other's degraded reads, however, then"−2.

"With a probability of p1, if k+1 data nodes are eliminated, the remaining n-1 nodes will have their burden reduced by (1 x) (k 1) p1o. That will, on average, lower the general read arrival rate at each of the remaining n 1 nodes by 1 (1 x) (k 1) p1o.The average general read arrival rate of the n k parity nodes may also be calculated in a similar fashion.

λpJ = (1 − x) (k − 1) poo + (1 x1) po1 — (1 x) k p p .For n 1, po and p are the same as in (3.8), thus n 1. Since the parity node may connect the degraded reads of all the data nodes, the first term in Eq. (3.9) includes all the other k 1 data nodes other than the hot data node. This is analogous to the second term in Eq. (3.8), except that the parity node is able to do this. Eq. (3.9)'s final two terms are identical to those of Eq. (3.8). Lemma 3.1: Assume that the direct read arrival rate of a single data node in a codeword is 1 when it becomes hot. If x1 xoJ, then x1J = xoJ may be achieved by load balancing by setting the probability of the hot data node joining the degraded reads of the other data nodes to p = (n1)d, where d = x( ) + (1 x)(k 2)p + (1 x)(k1)no (1 x)p1.Cording to Lemma 3.1, it is assumed that the proportion x of direct read jobs performed by the hot data node is the same as that performed by the other data nodes. By only modifying p1 to decrease the likelihood of the hot data node merging with the degraded reads of other data nodes, we may accomplish load balancing. If x1 oJ, then the proportion of direct read jobs for the hot data node can remain constant at x; otherwise, the fraction must be adjusted.

If we assume that the direct read arrival rate of a hot data node in a codeword is 1, then Lemma 3.2 holds. Setting the proportion of direct read jobs for the hot data node to x1 = e, where e = xo + (1 x) (k 2) poo + p1, will accomplish load balancing with 1J = oJ if x1 > oJ. p1 = 0 will prevent the hot data node from joining the degraded reads of the other data nodes.By modifying x, we can achieve load balancing over all of the nodes, including the parity ones, as discussed in Lemma 3.1 and 3.2. Load balancing's ability to cut down on delay is weakened since x gets small and the workload, and hence the bandwidth cost, of each node increases dramatically. It might potentially make things more sluggish. Since this is the case, we restrict our efforts to Lemmas 3.1 and 3.2, where load balancing is solely concerned with data nodes. Lemma 3.1 and 3.2 are so simple that their

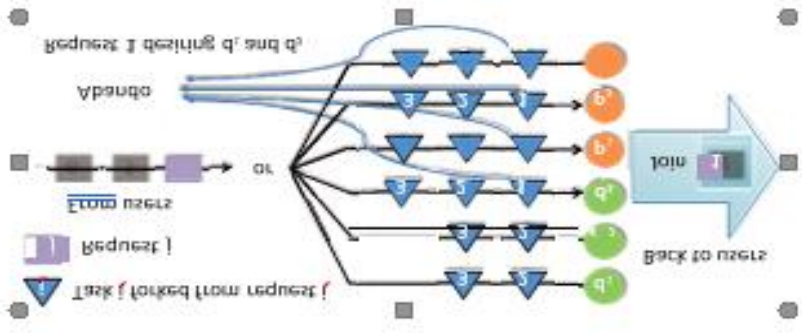
proofs are unnecessary here.In the next section, we suggest a method to cut down on wait times called Degraded Reads and Load Balancing (DRALB). As illustrated in Algorithm 3.1, the threshold value and the adjustment period T are both flexible and may be set to meet a variety of application-specific requirements. We employ Lemmas 3.1 and 3.2 to provide load balancing of data nodes, which, as can be seen from Algorithm 3.1, helps us minimise latency. The latency of hot data will be drastically reduced since burden is shifted away from the node producing the data.\

8. Research Design

For (n, k) MDS-coded storage systemreadinghe Redundant Request Technique, often known as RedS, has become an increasingly popular read technique in recent years. RedS may divide a codeword into n tasks and distribute them to each of the n nodes, regardless of the number of files that are requested by a read request inside the codeword. When k nodes out of n have completed delivering their services, the request is regarded to be done, and the remaining n k tasks are stopped as soon as possible.

The solution that we have built is based on RedS, and it has the ability to handle requests for files of varied sizes while simultaneously minimising access latency. (FRedS) is the acronym that we use to refer to this Flexible Redundant Scheme.

9. Conceptual Framework of the Study



10.Data Analysis

To answer your question in a general sense, "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)." It is possible to compute the fixed size M by making use of the storage space that is accessible at each node as well as the parameters of either HTSC (D, h) or FH FH HTSC. (D, h). Most of the time, users are only interested in a portion of the uncoded systematic component of a file, which is stored in one of the K nodes. Studies conducted in the past made the assumption that readers would want to have access to the whole contents of the use of a codeword to define the K-tree is a significant change, taking into consideration the fact that every single piece of information is now kept in the K-tree. On the other hand, this simplifies things too much and does not really represent reality. As an example, erasure coding is only accessible in WAS for files that exceed a certain size barrier, which might be as high as three gigabytes. The majority of individuals only use a tiny portion of the three gigabytes that are accessible, therefore it should not come as a surprise that it is a waste. This is in accordance with the functionality that was envisioned for the HTSC (D, h). Due to this reason, we will be focusing our attention on reading requests from customers who are only interested in a portion of the information that is stored in one of the K data nodes. "Inferences taken from this study,

Discussion

Our mathematical methodology in this research allows us to examine how request-specific file sizes affect the latency performance of replication and encoding. We provide a basic description of the latency-cost tradeoffs and propose two reading algorithms, FRedS and FRanS. When working with a varied dataset consisting of 88 samples, we additionally investigate the efficiency of coding and replication in relation to latency. Through comprehensive simulations using actual service "time traces from Amazon S3", we show the impact of storage cost, system load, cancellation cost, and non-uniform data popularity on delay performance. We also provide qualitative observations to support this connection.

Data and genetic material. Our comparison of coding and replication latency performance under the same storage cost reveals that, contrary to earlier findings, there are numerous factors to consider, the most important of which is whether the data popularity is uniform or not, making it difficult to draw any firm conclusions after considering real-world constraints.

Conclusion

To answer your question in a general sense, "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 parameters of HTSC (D, h) or FH FH HTSC may be used to determine the fixed size M. This can be done by utilising the available storage space at each node during the calculation. That is, D and H. Most of the time, users are only interested in a portion of the uncoded systematic component of a file, which is stored in one of the K nodes. Studies that were conducted in the past assumed that readers would want to have access to the whole 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." On the other hand, this simplifies things too much and does not really represent reality. As an example, erasure coding is only accessible in WAS for files that exceed a certain size barrier, which might be as high as three gigabytes. The majority of individuals only use a tiny portion of the three gigabytes that are accessible, therefore it should not come as a surprise that it is a waste. This is in accordance with the functionality that was envisioned for the HTSC (D, h). Due to this reason, we will be focusing our attention on reading requests from customers who are only interested in a portion of the information that is stored in one of the K data nodes. "Inferences drawn from this research"

11.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, 2019, pp. 29–43.

[2] M. Foley, “High availability HDFS,” in 28th IEEE Conference on Massive Data Storage, MSST, vol. 12, 2018.

[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, 2017, 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), 2018, 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, 2017.

[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, 2019.

[7] A. Fikes, “Storage architecture and challenges,” Talk at the Faculty Summit, 2018.

[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, 2019, 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, 2018. [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 productmatrix construction,” IEEE Transactions on Information Theory, vol. 57, no. 8, pp. 5227–5239, 2017.

[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,” 2018.

[12] N. B. Shah, K. Rashmi, P. V. Kumar, and K. Ramchandran, “Explicit codes minimizing repair bandwidth for distributed storage,” in 93 Information Theory Workshop (ITW), 2010 IEEE. IEEE, 2019, 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,

2018.

[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, 2017.

[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, 2019, pp. 376–384.

[16] A. Duminuco and E. W. Biersack, “Hierarchical codes: A flexible tradeoff for erasure codes in peer-to-peer storage systems,” Peer-to-peer Networking and Applications, vol. 3, no. 1, pp. 52–66, 2018.

[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, 2019, pp. 325–336.

[18] J. Li and B. Li, “Erasure coding for cloud storage systems: A survey,” Tsinghua Science and Technology, vol. 18, no. 3, pp. 259–272, 2018.

[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, 2017.

[20] A. Rudra, P. K. Dubey, C. S. Jutla, V. Kumar, J. R. Rao, and P. Rohatgi, “Ef- ficient Rijndael encryption implementation with composite field arithmetic,” in Cryptographic Hardware and Embedded Systems?CHES

2019. Springer, 2019, pp. 171–184.

[21] J. Brutlag, “Speed matters for Google web search,” Google. June, 2019.

[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), 2018, 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, 2018, 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, 2019, 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, 2018. 94

[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, 2018.

[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, 2018.

[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, 2019, pp. 287–300.

[30] A. Kala Karun and K. Chitharanjan, “A review on hadoophdfs infrastructure extensions,” in Conference on Information & Communication Technologies (ICT). IEEE, 2018, pp. 132–137.

[31] M. Harchol-Balter, Performance Modeling and Design of Computer Systems: Queueing Theory in Action. Cambridge University Press, 2019.
[32] H. A. David and H. N. Nagaraja, Order statistics. Wiley Online Library, 2019..
[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, 2019.
[34] S. B. Wicker and V. K. Bhargava, Reed-Solomon codes and them applications. John Wiley & Sons, 2018.
[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, 2019.
[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, 2019.
[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, 2017.
[38] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, and S. Sankar, "Row-diagonal parity for double disk failure correction," in Pro- ceedings of the 3rd USENIX Conference on File and Storage
Technologies, 2019, pp. 1–14.
[39] C. Huang and L. Xu, "Star: An efficient coding scheme for correcting triple storage node failures," IEEE Transactions on Computers, vol. 57, 95 no. 7, pp. 889–901, 2018.
[40] J. L. Hafner, "Weaver codes: Highly fault tolerant erasure codes for storage systems." in FAST, vol. 5, 2019, pp. 16–16.
[41] L. E. Dickson, Linear groups: With an exposition of the Galois field theory. Courier Dover Publications, 2018.
[42] K. M. Greenan, X. Li, and J. J. Wylie, "Flat XOR-based erasure codes in storage systems: Constructions, efficient recovery, and tradeoffs," in Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on. IEEE, 2019, pp. 1–14.
[43] L. Kleinrock, Queueing Systems: Volume 2: Computer Applications. John Wiley & Sons New York, 1976, vol. 82.
[44] D. Borthakur, R. Schmidt, R. Vadali, S. Chen, and P. Kling, "HDFS RAID," in Hadoop User Group Meeting, 2016.
[45] Q. Shuai and V. O. K. Li, "A general model of latency performance in dis- tributed storage systems," Department of Electrical and Electronic Engineer- ing, The University of Hong Kong, Tech. Rep. No.TR-2015-03, April 2019.
[46] O. Khan, R. Burns, J. Plank, W. Pierce, and C. Huang, "Rethinking erasure codes for cloud file systems: Minimizing I/O for recovery and degraded reads," in Proc. of USENIX FAST, 2018.
[47] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in Mass Storage Systems and Technologies (MSST), 2018 IEEE 26th Symposium on. IEEE, 2019, pp. 1–10.
[48] K. Gardner, S. Zbarsky, S. Doroudi, M. Harchol-Balter, and E. Hyytia, "Re- ducing latency via redundant requests: Exact analysis," in Proceedings of the 2015 SIGMETRICS International Conference on Measurement and Modeling of Computer Systems. ACM, 2019, pp. 347– 360.
[49] Q. Shuai and V. O. Li, "Delay performance of direct reads in distributed stor- age systems with coding," in 17th International Conference on High Perfor- mance Computing and Communications (HPCC). IEEE, 2017, pp. 184–189.
[50] G. Joshi, E. Soljanin, and G. Wornell, "Queues with redundancy: Latency- cost analysis," ACM SIGMETRICS Performance Evaluation Review, vol. 43, no. 2, pp. 54–56, 2016.
[51] S. Chen, Y. Sun, U. C. Kozat, L. Huang, P. Sinha, G. Liang, X. Liu, and N. B. Shroff, "When queueing meets coding: Optimal-latency data retrieve scheme in storage clouds," in Proceedings of INFOCOM. IEEE, 2018, pp. 1042–1050.
[52] S. Kadhe, E. Soljanin, and A. Sprintson, "Analyzing the download time of availability codes," in International Symposium on Information Theory 96 (ISIT). IEEE, 2016, pp. 1467–1471.
[53] R. Nelson and A. N. Tantawi, "Approximate analysis of fork/join synchroniza- tion in parallel queues," IEEE Transactions on Computers, vol. 37, no. 6, pp. 739–743, 2019.
[54] E. Varki, A. Merchant, and H. Chen, "The m/m/1 fork-join queue with variable sub-tasks," Unpublished, available online, 2018.
[55] G. Joshi, Y. Liu, and E. Soljanin, "Coding for fast content download," in the 50th Annual Allerton Conference on Communication, Control, and Comput- ing. IEEE, 2019, pp. 326–333.
[56] P. J. Bickel et al., "Some contributions to the theory of order statistics," in Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics. The Regents of the University of California, 2016.
[57] G. Joshi, E. Soljanin, and G. Wornell, "Efficient redundancy techniques for latency reduction in cloud systems," arXiv preprint arXiv:1508.03599, 2019.
[58] N. Papadatos, "Maximum variance of order statistics," Annals of the Institute of Statistical Mathematics, vol. 47, no. 1, pp. 185–193, 2016.
[59] G. Szekely and T. Mori, "An extremal property of rectangular distributions," Statistics & probability letters, vol. 3, no. 2, pp. 107–109, 2019.
[60] W. Hu¨ rlimann, "Generalized algebraic bounds on order statistics functions, with application to reinsurance and catastrophe risk," in Proceedings of the 31st International ASTIN Colloquium, Porto Cervo,
2018, pp. 115–129.
[61] D. Bertsimas, K. Natarajan, and C.-P. Teo, "Tight bounds on expected order statistics," Probability in the Engineering and Informational Sciences, vol. 20, no. 04, pp. 667–686, 2016.
[62] R. Rodrigues and B. Liskov, "High availability in DHTs: Erasure coding vs. replication," in Peer-to-Peer Systems IV. Springer, 2019, pp. 226– 239.
[63] S. Nath, H. Yu, P. B. Gibbons, and S. Seshan, "Subtleties in tolerating corre- lated failures in wide-area storage systems." in NSDI, vol. 6, 2016, pp. 225– 238.
[64] P. Gopalan, C. Huang, H. Simitci, and S. Yekhanin, "On the locality of code- word symbols," 2019.

DOI: https://doi.org/10.15379/ijmst.v10i5.3791