ISSN:0975-9646

A Survey on Erasure Coding Techniques for Cloud Storage System

Kiran P. Pawar [#1], R. M. Jogdand [*2]

[#1] *M.Tech, Department of Computer Science, Gogte Institute of technology, Udyambag Belagavi, Karnataka, India*

[*2] *Professor, Department of Computer Science ,Gogte Institute of technology, Udyambag Belagavi, Karnataka, India*

***Abstract*— In the present period of cloud computing, information look up in the cloud is being produced at a huge rate, and in this way the cloud storage framework has gained to be one of the key segments in cloud computing. By putting away a huge amount of information in commodity disks in data centres that moderates the cloud, the cloud storage framework must think of one as question precisely: how would we store information dependably with a high efficiency as far as both storage overhead and data integrity? Despite the fact that it is easy to store repeated information to tolerate some amount of data losses, it experiences a low storage efficiency. Traditional erasure coding techniques for example, Reed-Solomon codes, have the capacity to accomplish a much lower storage expense with the same level of resilience against disks failures. Nonetheless, it brings about much higher repair costs, also a significantly higher access latency. In this sense, planning new coding techniques for cloud storage has picked up a significant measure of consideration in both the scholarly world and the industry. In this paper, we analyze the current effects of coding techniques for cloud storage. Specifically, we exhibit these coding techniques into two classifications: regenerating codes and locally repairable codes. These two types of codes meet the necessities of cloud storage along two unique axes: reducing bandwidth and I/O overhead. We show a review of recent advances in these two types of coding techniques. In addition, we present the principle thoughts of some specific coding techniques at high state, and examine their inspirations and performance.**

***Keywords*— erasure coding; cloud storage; regenerating codes; locally repairable codes.**

I. Introduction

With the less use of bandwidth and storage expenses, a direction has been observed that driving IT companies, for example, Google, Microsoft, and Amazon, create their services inside data centres and allow those services to use globally using a high-bandwidth network. This new technology of giving services is called *cloud computing.*

In the era of cloud computing, storage has not just been an essential segment of substantial scale cloud services, additionally been given as a virtual storage infrastructure in a pay-as-you-go way, for example, Elastic Block Storage (EBS) provided by Amazon [1].

To meet the prerequisites of the huge volume of storage, the cloud storage system needs to scale out, i.e., putting away information in large disks. In this sense, it turns into a big challenge for cloud storage to keep up data integrity, because of both the huge number of disks and their commodity nature. Despite the fact that the chances of disk failure is a little part inside the data centres, there can in any case be a substantial number of such failure regular because of the more number of disks. For example [2], in a Facebook cluster with 3000 hubs, there are commonly no less than 20 repairs activated regular. Aside from storage components , alternate systems in the data centre, for example, the networking or power systems, may bring the question of data unavailability in the data centre.

To manage the data loss and to ensure the integrity of data which is stored in the cloud, it is simple way to store replicas of data in more than one disk, such that data losses can be handled till there is no less than one replica accessible. Then again, replicas can significantly lessen the storage efficiency. For instance, if data are stored in a 3-way replication, the effective storage can be no superior to anything like $\frac{1}{3}$ of the total storage.

In the event that the design principle is to expand the storage efficiency without sacrificing the power to handle disk failures, the storage system must be able to store encoded data by erasure coding. Prior to the rise of cloud computing, erasure coding has long been proposed to correct errors in communication system or storage or to delete errors. For instance, RAID 6 can make up for at most 2 disk failures using parity coding, whose storage efficiency is at most $1 - \frac{2}{n}$ where n is the number of disks in total. Reed-Solomon codes can give much more flexibility such that any number of disk failures under a specific threshold can be handled. An online secure storage as a service, utilizes Reed-Solomon codes to guarantee data integrity, by encoding data using Reed Solomon codes after doing the encryption of data.

Nonetheless, two principle disadvantages traditionally keep erasure coding from being practical and prominent in cloud storage. Initially, to write or read data, the system has to encode or decode data, prompting a high access latency and low access throughput because of the CPU limit. Second, however erasure coding stores information as numerous coded blocks, when one coded block gets lost, the system must get to various coded blocks that are sufficient to recover all the information. It is evaluated that regardless of the fact that half of the information in the

Facebook cluster are encoded, the repair traffic will saturate the system such as network links in the cluster [2]. This overhead makes the repair with erasure coding exceptionally costly as far as both bandwidth for data transfer and overhead of disk I/O. unfortunately, applications facilitated in the cloud are more sensitive to the performance of disk I/O ,bandwidth is dependably a constrained asset inside the data centres subsequent to most data centres present connection over subscription.

To meet the prerequisites of cloud applications for storing purpose, the configuration of new coding methods for cloud storage has pulled in a generous measure of enthusiasm for the research group. In this paper, we look at recent advances of erasure coding techniques in the connection of cloud storage. Specifically, we present coding techniques that improve the data repair overhead in two unique axes of design objectives: reducing the bandwidth consumption and optimizing overhead of disk I/O. because of the high frequency of repairing data and the high repair of conventional erasure coding overhead, advancements in both of these two axes fall into the considerations to repair data.

In the point of view of bandwidth utilization, Dimakis et al. [3] proposed a group of regenerating codes. Taking into account a model motivated by network coding, the repairs of data which has coded can be displayed into a data flow chart, with an imperative that keeps up the capacity to handle disk failures. In light of this model, an ideal trade-off between bandwidth and storage can be derived, such that given the measure of information which is stored on the disks, we can acquire an ideal lower bound of the measure of information that ought to be exchanged at the time of repair. In the trade-off curve, two great focuses draw in substantially more consideration than inside focuses, comparing to the base storage cost and the base bandwidth cost, respectively. To accomplish these two great focuses, number of papers proposed occurrences of regenerating codes that either handles the disk failures using randomised codes , or utilizing inference alignment to build deterministic codes . Using deterministic coding, lost data can be repaired precisely, recommending that we can develop systematic codes to accomplish an access latency as low as replicas. In addition, cooperative regenerating codes make it conceivable to repair from various disk failures with an even lower bound of bandwidth utilization, with the help of cooperation among participating servers.

Regenerating codes spares bandwidth by not sending information that are pointless to the specific newcomer. It is required that the information sent out of suppliers must take the relating information(i.e., the entropy) required by the newcomer, with a specific end goal to handle disk failures. Therefore, just with the exception of some uncommon cases, most occurrences of regenerating codes need to ask suppliers to send linear combinations of data from their information to the newcomer. At the end , regenerating codes cannot save the disk I/O but they only can save bandwidth. compared with traditional erasure coding, disk I/O will even most likely be expanded with regenerating codes. With regenerating codes, the inevitable amount of data read from disks at the time of repair is as yet going to be significantly more than the measure of information kept in touch with the newcomer. The excessive operations of disk I/O can significantly influence the performance of general disk I/O in the data centre.

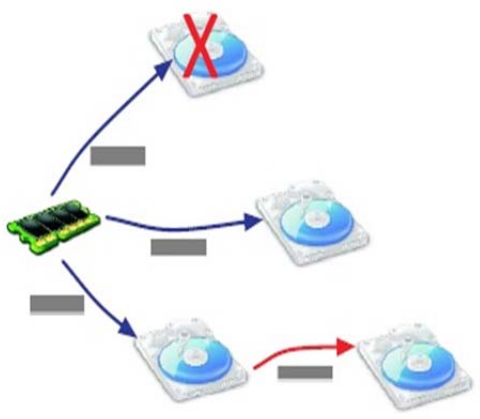
In this sense, a few groups of coding techniques have been proposed to decrease disk I/O overhead at the time of repair. Dissimilar to regenerating codes that spare the bandwidth utilization by causing more disk I/O overhead, every one of them feature a smaller number of disk accessed in the repair. This feature is accomplished by implementing that data in one disk must be repaired by data in some certain disk. This thought will handle disk failures, however significantly decrease the number of disks to be visited and number of bits to be read. In this paper, we present three representative proposals: hierarchical codes, self-repairing codes, and simple regenerating codes. Also, we demonstrate the major exchange off between disk failures and disk I/O overhead.

II. Erasure Coding And Its Performance Metrics

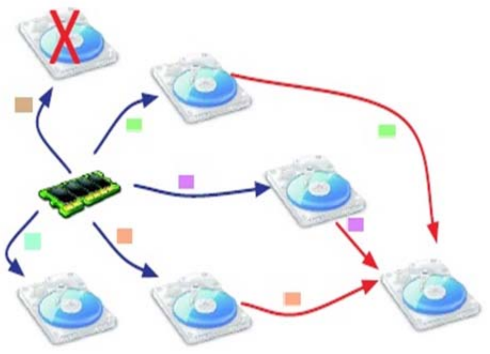
1. Erasure coding in storage systems

In a storage system where information are stored on a more number of commodity disks, it is clear that disk failures cannot be considered as just special cases, but rather generally speaking. Along these lines, the storage system needs to store redundancy such that when a specific number of disks lose data, data can in any case be available from different disks. For instance, the *N*-way replication, which stores *N* replicas in *N* different disks, has the capacity to handle at most *N*- 1 disk failures. Figure 1a delineates a sample of 3-way replication, where the first information are spread into 3 distinct disks and any one disk has the capacity to repair or access the original data. However, *N*-way replication can just accomplish a storage efficiency of $\frac{1}{N}$ best case scenario. Erasure coding, on the other hand, is able to handle the same number of disk failures, yet with a vastly improved storage efficiency. Among erasure coding, Maximum Distance Separable (MDS) codes (e.g., Reed-Solomon codes [4]) accomplish the ideal storage efficiency.

Assume that in the storage system, data are sorted out in the unit of data object, which may relate to a file or a fix-sized block in diverse storage system. Accept that an data objects can be put away onto n disks. Given k as arbitrary number , where $k < n$, (n, k) MDS codes can promise to handle at most *n-k* disk failures, i.e., *k* disks are sufficient to get to any piece of the first data. Specifically, the data object is encoded into *n* coded pieces and are consistently dispersed into the *n* disks.



(a) 3-way replication



(b) (5,3) MDS codes

Dissemination
Repair
Original data object
Coded data

Fig. 1 Comparison of 3-way replication and (5,3) MDS codes.

Assume the data object size is M bits, the size of each coded piece ought to be $\frac{M}{k}$ bits, on the off chance that we don't consider the information's metadata object. In this sense, the storage efficiency of MDS codes is, best case scenario $\frac{k}{n}$. Contrasted with the 3-way replication as appeared in Fig. 1b, (5,3)MDS codes can at present handle at most 2 disk failures, while enhancing the storage efficiency by 80%.

To get to the data object access, the system needs to get to k distinctive coded blocks (from k diverse blocks) and use MDS codes for decoding algorithm to recover the first data object. Apparently, the decoding algorithm causes an extra access latency. To enhance the access latency, the storage system can utilize a cache to store one replica of the first data object as well [5].

Recovering the entire data object for data access may sound sensible in most cases. however, from the storage system perspective, absolutely superfluous to recover the entire data object in the event that we just need to repair one lost coded block, as what we need is only a little piece of the data object. Tragically, before the presence of regenerating codes, all MDS codes, to our best information, require to access in any event k disks to repair even stand out disk, while on account of replication, to repair one replica we just need to exchange one replica. This prerequisite can drastically expand both disk I/O and bandwidth overhead in a data centre and significantly influence the storage system performance and different applications facilitated in the cloud.

2. *Performance metrics*

It has been clarified that it is insufficient to consider just the storage efficiency and the handling against disk failures in cloud storage systems. The performance metric that ought to be considered when planning erasure coding for cloud storage ought to likewise include:

Repair bandwidth. To repair a failed disk, data put away on that disk ought to be repaired in a substitution disk. The server with the substitution disk, called a newcomer, needs to recover information from some current disks. In the event that the servers that host these current disks, called providers, send out coded blocks of raw data , the bandwidth used to transmit the current coded blocks equals the size of these coded blocks and then the newcomer encodes the received data by itself to create the data which is lost. Not with standing, if encoding operations can be performed both on the newcomer and suppliers instead of on the newcomer just, a much littler size of data can be exchanged. As appeared in Fig. 2, if storage nodes stores data are encoded by vector codes, such that each coded block contains more than one coded fragments, the bandwidth utilization can be spared when suppliers sends linear combinations of their coded segments during the repair.

In the spearheading paper of Dimakis et al. [3], an astonishing and promising result was demonstrated that the bandwidth utilized as a part of the repair can be roughly as low as the size of the repaired block by encoding operations of suppliers and the group of erasure codes that accomplishes the ideal bandwidth utilization during the repair was called *regenerating codes.*

Repair I/O. Other than bandwidth, another performance metric in the repair for erasure coding is disk I/O at the partaking servers. Specifically, the writing operations are performed just at the new comer, and the measure of information written ought to approach the span of the coded blocks. As the written work operations are unavoidable, what we truly care is really the measure of information read from disks of suppliers.

Like the data bandwidth utilization, traditional erasure codes will request suppliers to peruse k blocks altogether just to repair one block. As appeared in Fig. 2, encoding operations on suppliers cannot decrease the amount of information read, but then just diminish the amount of information exchanged to the newcomer, since the fragments conveyed of suppliers are encoded from all coded sections in suppliers, which all must be perused from disks. Hence, new coding systems should be proposed to enhance the disk I/O at the time of repair.

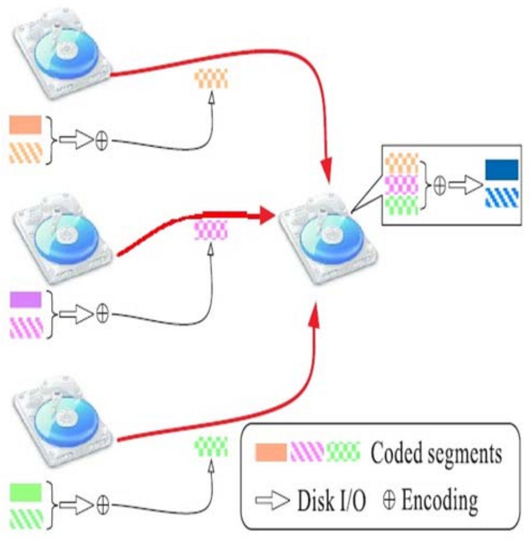


Fig. 2 Suppose that any two storage nodes suffice to recover the first data(k=2),where every supplier stores one coded block including two coded portions.

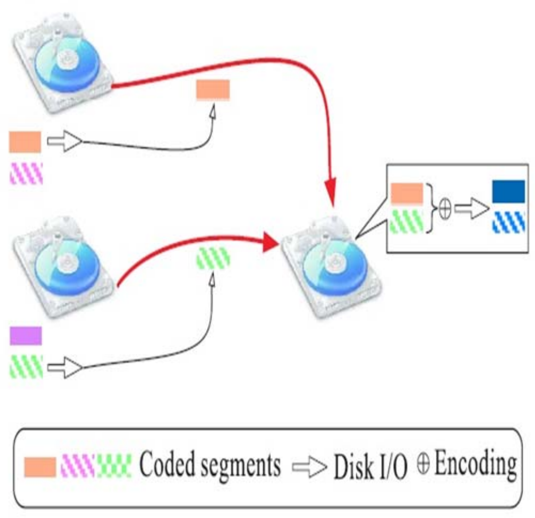


Fig. 3 The disk I/O amid the repair can be spared by getting to specific coded sections or/and from specific suppliers.

In Fig. 3,we present two illustrations of methods that can be utilized to develop erasure codes with low disk I/O at the time of the repair. One conceivable path is to acquire specific coded blocks from suppliers, and consequently other coded fragments in the same supplier won't be encoded and subsequently won't be perused. Another strategy is to get to specific storage nodes as suppliers, as opposed to getting to any k storage nodes. A few groups of erasure codes have been proposed such that when one storage node fails, the new comer must get to information from a little number of specific storage nodes.

Access latency. Because of the decoding operations, the access latency of information encoded by erasure coding must be much bigger than replicas. In any case, efficient codes, in which the first information can be inserted into code blocks, have the capacity to keep up a higher storage efficiency than replicas while accomplishing a smaller access latency, since we now just need to get to the orderly part with no decoding operations. This interesting property of efficient codes accompanies a high value that makes the repair a great deal more mind boggling than non-efficient codes, on the grounds that the repaired information ought to be the very same as the lost information, at any rate for the embedded unique information. In this way, the deterministic development of codes must be considered when outlining efficient erasure codes for cloud storage.

Storage efficiency. The storage efficiency indicates the ratio of the first information to the real amount of information put away on disks. in other words, given the same amount of information, with a higher storage efficiency we can utilize a littler amount of space of storage to handle the same number of disk failures. MDS codes accomplish the ideal storage efficiency, i.e., given n and k, MDS codes can be built such that any k among all n coded blocks are sufficient to recover the first information. For instance, in Fig. 1, to handle two disk failures, $3M$ bits must be put away with replications, while just $\frac{5M}{3}$ bits are required to store with MDS codes. On the other hand, in the event that we decide to hold this property, it is unavoidable to bring about high disk overhead at the time of repair [2]. Because of the diminishing costs of large volume disks, the necessity for storage efficiency can be casual such that a much littler number of disks are required to visit in the repair.

In this paper, we concentrate on the first two metrics, which are unequivocally identified with the repair in the storage system. Specifically, in Section3 we show the coding procedure, called regenerating codes, that has the capacity to accomplish the ideal bandwidth bound in the repair. In Section 4, we audit the coding methods along the axis of lessening the disk I/O in the repair. For every coding system, we talk about their performance in the other two metrics along the way.

III. Tradeoffs Between Storage And Bandwidth: Regenerating Codes

1. *Information flow graph*

The trade-off in the middle to research the bandwidth utilization in repairs with erasure coding, Dimakis et al. [3] proposed to utilize the data flow diagram, which is an tool utilized as a part of the network coding analysis, as a model to describe the storage and bandwidth.

As appeared in Fig. 4, in the data flow chart, all servers can be classified as the source, storage nodes, and the data collector. The source signifies the server where the information object is started. Assume that the information object size is M bits. In the wake of encoding, coded blocks of α bits are scattered into n storage nodes. Particularly, the source is represented by a vertex and a storage node is shown by two vertices in the data flow diagram. The amount of data stored in storage node is represented by weight of the edge. Subsequently, after the spread, all n storage nodes store α bits and any k of them suffice to recover the original data object, suggesting that $k\,\alpha \geq M$. A virtual vertex called information collector has the capacity connect with any k storage nodes to recover the first information.

At the point when a storage node fails, a newcomer does not interface with k accessible storage nodes, as well as d storage nodes as suppliers ($d \geq k$). Not quite the same as

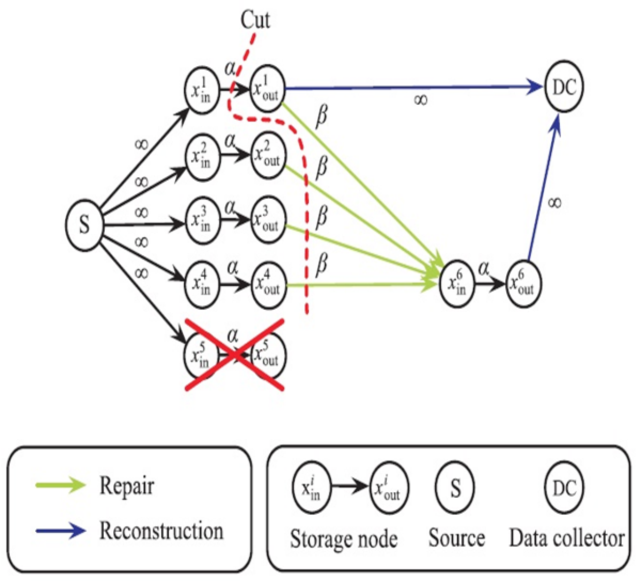


Fig. 4 A delineation of the data flow chart that relates to(4,2)MDS codes. This figure was firstly appeared in Ref. [3].

traditional MDS codes, the newcomer can get β bits from every supplier, $\beta \leq \alpha$, shown by the edge's heaviness between the supplier and the newcomer. Subsequent to accepting a sum of $r = d\,\beta$ bits from suppliers, the newcomer stores α bits and turns into another storage node. It is required that after any rounds of repairs, the MDS property that any k coded blocks suffice to recover the first information object dependably holds. In other words, the information collector can associate with not just the storage nodes that get information specifically from the source, yet the storage nodes repaired from newcomers in repairs too. Additionally, the weights of edges joined to the source or the information collector are all set to be infinity.

Along these lines, the repair issue in the storage system can be deciphered as a multicast issue in the data flow chart, where the source is multicasting information to all conceivable information collectors. It is understood in the multicast issue that the most extreme multicast rate approaches the base cut separating the source from any receivers, and this rate can be accomplished through network coding [6]. Accordingly, it can be demonstrated that the MDS property holds after any rounds of repairs, in the event that them in-cut in the data flow diagram is no less that the size of first information object. By computing the base min-cut in the data flow diagram, given d and β we can determine the base estimation of α, and afterward we acquire the trade-off curve in the middle of α and the aggregate bandwidth utilization r.

2. Minimum-storage regenerating codes and minimum-bandwidth regenerating codes

The codes that accomplish that the min-cut in the data flow diagram measures up to the information object is called *regenerating codes*. Given the trade off between bandwidth r and storage α , two unique instances of regenerating codes interest us most, which compare to the base storage space required at storage nodes and the base aggregate bandwidth utilization in the repair, respectively.

The regenerating codes that accomplish the base storage in storage nodes is called *Minimum Storage Regenerating* (MSR) codes, where

$$(\alpha_{\text{MSR}}, r_{\text{MSR}}) = \left(\frac{M}{k}, \frac{Md}{k(d-k+1)}\right) \qquad (1)$$

Notice that in above equation (1) the α equivalents to the size of coded blocks of traditional MDS codes, and in this manner MSR codes can be viewed as MDS codes. Then again, since

$$r = \frac{Md}{k(d-k+1)} \longrightarrow \frac{M}{k} \text{ as } d\longrightarrow\infty, \qquad (2)$$

MSR codes expend bandwidth in the repair around the same as the amount of one coded block, while traditional MDS codes use bandwidth that equivalents the size of k coded blocks. In this sense, MSR codes spare a significant measure of bandwidth in the repair.

The other great focuses in the trade-off in the middle of storage and bandwidth is called *Minimum Bandwidth Regenerating* (MBR) codes, where

$$(\alpha_{\text{MBR}}, r_{\text{MBR}})=\left(\frac{2Md}{k(2d-k+1)}, \frac{2Md}{k(2d-k+1)}\right) \qquad (3)$$

In above equation (3) MSR codes accomplish the base bandwidth utilization among regenerating codes. Despite the fact that MBR codes require more storage than MSR codes, the newcomer just needs to get precisely the measure of information it needs to repair, and the bandwidth and storage both join to $\frac{M}{k}$ when d is sufficiently large.

To actualize regenerating codes, the least difficult way (however not as a matter of course the most efficient way) is to utilize random linear coding [7] which is motivated by network coding. Partitioning the first information object into k blocks, a coded block created by random linear coding is an random linear combination of the k blocks. The encoding operations are performed on a Galois field GF(2^q). As a Galois field size of 2^8 relates to a byte in the PC, q is generally set to be an essential times of 8 to make encoding operations advantageous. Given any k coded blocks and their coding coefficients, the decoding operations are simply solving corresponding linear system with k unknowns (i.e., unique blocks) and k equations. At the point when q is sufficiently large, for example, 16 or 32, any k coded blocks can be decoded with a high probability.

Then again, the randomized development of regenerating codes can experience the effects of a significant computational complexity nature, particularly when parameters are not legitimately selected [8]. In additional, by no methods the randomized regenerating codes can promise to repair information precisely as the lost information. Far and away more terrible, even an large Galois field cannot guarantee that any k coded blocks are decodable, however just with a high probability, because of the randomized coefficients. The repairs numbers, in any case, cannot be effortlessly constrained, proposing that after a substantial number of repairs information trustworthiness cannot be kept up, as the randomness accumulates step by step in coded blocks.

Therefore, it is important to find explicit development of regenerating codes, particularly for MSR and MBR codes. Further, it is require that the lost block can be repaired precisely given the explicit development.

3. *Exact regenerating codes*

The most vital device used to build exact regenerating codes is *interference* alignment, which is at first proposed for wireless communication. The fundamental thought of interference alignment is that the undesired vectors can be wiped out by adjusting them onto the same linear subspace. Figure 5 outlines how interference alignment serves to accomplish careful regenerating codes.

We assume that in Fig. 5 information are encoded by (n=4, k=2, d=3) MSR codes, i.e., any two of the four nodes can recover the first file. In every storage node, each coded block contains two coded fragments, for example, (A_1, A_2) in the storage node which has failed. To recover A_1 and A_2, the newcomer contacts 3 storage nodes as suppliers and downloads half of a coded block, i.e., a coded portion from every supplier to accomplish the bandwidth utilization of MSR codes. Notice that every supplier claims coded fragments containing parts of B_1 and B_2, which are undesirable to the newcomer. To delete B_1 and B_2, every supplier can send a fragment in which B_1 and B_2 are adjusted onto the same linear subspace of B_1+ B_2. Obviously, B_1+B_2 can be dispensed with as one obscure and A_1 and A_2 can be decoded by explaining three mathematical statements with three unknowns.

Concerning precise MBR codes, Rashmi et al. [9] proposed a Product-Matrix development which has the capacity explicitly build (n ,k, d) precise MBR codes on a finite field of size n or higher with any decisions of (n, k, d) if $k \le d \le n$. The Product-Matrix development produces vector MBR codes such that a coded blocks contains various coded fragments, much the same as the illustration appeared in Fig.5.

With respect to correct MSR codes, the decisions of parameters turn out to be more complicated. Suh and Ramchandran [10] proposed an explicit development of scalar correct MSR codes where $d \ge 2k$ - 1, over a finite field of size no less than 2(n - k). Rashmi et al. [9] enhanced the decisions of parameters such that $d \ge 2k$ - 2, by developing correct MSR codes utilizing the Product-Matrix development, with a bigger finite field of size at any rate n(d- k + 1). In Ref. [11], Shah et al. had demonstrated that no scalar correct MSR codes exist when $d < 2k$ - 3.

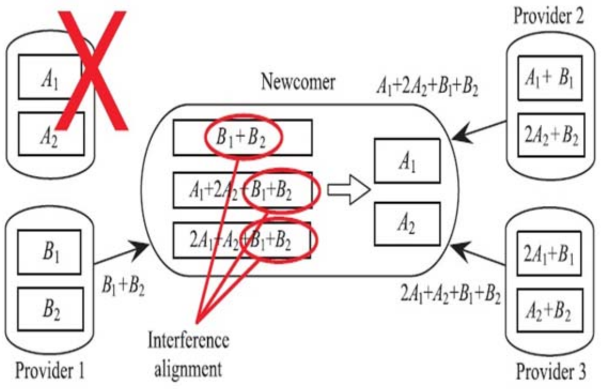


Fig. 5 A case of (4,2,3) precise MSR codes.

Cadambe et al. [12] and Suh and Ramchandran [13] demonstrated that any decisions of (n, k, d)could be accomplished asymptotically when building vector correct MSR codes, as the field size goes to infinity. Developing regenerating codes on an large finite field is illogical because of its overwhelming encoding/decoding overhead. Papailiopoulos and Dimakis [14] demonstrated that (n=k + 2, k, d = k +1) vector correct MSR codes can be developed explicitly, consolidating interference alignment and Hadamard matrices.

Since the correct repair makes much more sense for the systematic part than the parity part of exact regenerating codes, we can build hybrid regenerating codes that just support the correct repair of the systematic part, while the parity part is still repaired by the functional repair. Wu [15] developed the (n, k, d = k+1) hybrid vector MSR codes when $2k \le n$, where the field size is more prominent than 2 $\binom{2n-1}{2k-1}$. Tamo et al. [16] and Cadambe et al. [17] both proposed (n, k, d =n-1) hybrid MSR codes for arbitrary k and n.

4. *Cooperative regenerating codes*

In some storage systems, for example, Total Recall [18], keeping in mind the end goal to prevent unnecessary repairs caused by temporary node departures, the system repairs nodes which have failed in group when the quantity of failed node numbers achieves a threshold limit. Hu et al. [19] first found that if newcomers can collaborate, there exist cooperative regenerating codes that accomplish a superior trade-off curve in the middle of bandwidth and storage. As yet examining the min-cut in the data flow diagram, Hu et al. [19] demonstrated that (n, k, $d = n\text{-}t$, t)randomized *Minimum Storage Cooperative Regenerating* (MSCR) codes can achieve the bandwidth utilization of $\frac{M}{k} \cdot \frac{n-1}{n-k}$ with t suppliers ($n\text{-}\ t \geq k$). Notice that the bandwidth utilization is free of the number of suppliers.

A more broad consequence of the bound of bandwidth utilization is demonstrated that per newcomer

$$(\alpha_{\mathrm{MSCR}}, \beta_{\mathrm{MSCR}}) = (\frac{M}{k}, \frac{M}{k}, \frac{d+t-1}{d+t-k}) \text{ [20] [21]} \qquad (4)$$

and

$$(\alpha_{\mathrm{MBCR}}, \beta_{\mathrm{MBCR}}) = (\frac{M}{k}, \frac{2d+t-1}{2d+t-k}, \frac{M}{k}, \frac{2d+t-1}{2d+t-k}) \text{ [21]} \quad (5)$$

ShumandHu [22] proposed an explicit development of (n, k, $d = k$, $t = n\text{-}\ k$) correct MBCR codes. Wang and Zhang [23] demonstrated that with respect to every single conceivable estimation of (n, k, d, t), there exist explicit developments of correct MBCR on a field of size at any rate n. Then again, when $d= k \neq n\text{-}t$, (n, k, d, t)scalar MSCR codes can be constructed [20]. Le Scouarnec [24] talked about the development of correct MSCR codes with some different decisions of parameters when $d \geq k = 2$. The presence of correct MSCR codes with different estimations of parameters still remains an open challenge.

5. *Repair-by-transfer regenerating codes*

The regenerating codes we have said above require suppliers to encode their information to adjust vectors in a specific linear subspace and the newcomer to encode got information to dispense with undesired parts. The arithmetic operations performed on a finite field can be costly, if the field size is vast. In this manner, the cloud storage system will support the *repair-by-transfer* property that in the repair no arithmetic operations are required at suppliers. With the repair-by-transfer property, the disk I/O overhead can be insignificant since just information required by the newcomer will be perused from suppliers. Additionally, the storage node then can have no insight, such that its usefulness can be actualized by implemented with a minimal cost.

A few decisions of parameters have been considered to build the corresponding repair-by-transfer regenerating codes. Shum and Hu [25] and Hu et al. [26] developed (n, k =2,d = n-1)and (n, k = n-2,d = n-1) functional repair-by-transfer MSR codes, individually. Then again, (n, k = n-2,d = n-1) correct MBR codes can be built over a finite field of size 2 [27]. The presence of correct regenerating codes remains an open issue with respect to most different decisions of parameters. The main known result is that if $d \geq 2$ and $t \geq 2$, any (n, k, d, t) linear correct MBCR codes cannot accomplish the repair-by-transfer property [23].

All things considered, Rashmi et al. [28] proposed an instinctive graph based development of repair-by-transfer correct MBR codes, where any missing block must be repaired from its immediate neighbours in the graph. Fractional repetition codes are utilized in the development such that immediate neighbours have the same fragment and the newcomer just needs to get replicas of the fragments from its neighbours to repair itself, where no arithmetic operations are required at suppliers, as well as at the newcomer also. It is demonstrated that any (n, k, d = n-1)MBR codes can be developed exceptionally with this technique, $n > k$.

Despite the fact that repair-by-transfer regenerating codes have the capacity to accomplish the base disk I/O overhead, just some specific decisions of parameters have examples of relating regenerating codes as such. By and by, cloud storage systems can have a wide range of parameter decisions because of their own necessities. Therefore, it is desirable that erasure coding can achieve low disk I/O overhead with an extensive variety of parameter decisions, regardless of the possibility that a few properties, for example, the MDS property or the storage efficiency, will be sacrificed.

6. *Regenerating codes for pipelined repair*

Aside from some specific decisions of parameters that have relating occurrences of repair-by-transfer regenerating codes, most cases of regenerating codes oblige suppliers to convey linear mixes of their coded blocks to the newcomer. As it were, suppliers need to peruse the greater part of their information despite the fact that the amount of information to be conveyed covers just a little part. In this sense, despite the fact that the bandwidth utilization can be diminished to the ideal by regenerating codes, the disk I/O increments with the number of suppliers. As the number of suppliers can be much bigger than k, the disk I/O overhead with regenerating codes will not be enhanced, however turn out to be much severer than MDS codes.

Clearly, as the disk I/O overhead is reliant with the number of suppliers at the time of the repair, the disk I/O can be spared if the number of suppliers is decreased while keeping up alternate properties of regenerating codes, particularly the bandwidth ideal utilization. To accomplish this objective, Li et al. [29] proposed a pipelined repair with minimum storage regenerating codes. The guideline of pipelined repair is that suppliers of successive newcomers are chosen to be overlapped, in light of the fact that any k or more than k distinctive suppliers are sufficient in the repair. Consequently, sequential repairs can be pipelined, as appeared in Fig. 6. Still, a newcomer joins when there is one disk failure, yet newcomers will not be completely repaired in only one however numerous rounds of repairs and distinctive newcomers can get and gather information from suppliers at the time of the repair. After every round of repair, the "eldest" newcomer will contact enough suppliers and "graduate" to become a storage node.

The most significant benefit of pipelined repair is the lessening of suppliers in the repair. For instance in Fig. 6, the number of suppliers can be decreased by 67%. Actually, the free (n, k, d)regenerating codes for the pipelined repair can diminish the number of taking part nodes to as few $2\sqrt{d+1}-1$ in which only $\sqrt{d+1}-1$ are suppliers, while as yet keeping up the same bandwidth utilization of (n ,k, d) minimum-storage regenerating codes. It is shocking to find that the number of suppliers can be even not as much as k, the length of the newcomer has the capacity get information from in any event d suppliers before its "graduation". The (n, k, d, t) cooperative pipelined regenerating codes, which can be viewed as a generalization of free pipelined regenerating codes, have been talked about in Ref. [29], which require $\sqrt{t(d+t)}-t$ suppliers while keeping up the bandwidth utilization of (n, k, d, t) cooperative regenerating codes.

The overhead brought by the pipelined repair is the storage space required by partially repaired newcomers, as newcomers must be completely repaired after different rounds of repairs. The extra storage overhead equals the storage space used by $\sqrt{(d+t)t}-t$ storage nodes. Subsequently, the storage efficiency will be lessen.

However, it can be demonstrated that this overhead is minor in practical cloud storage systems [29]. In addition, each block in completely repaired new comers can in any case keep up the MDS property, despite the fact that the number of suppliers in the repair can be not as much as k.

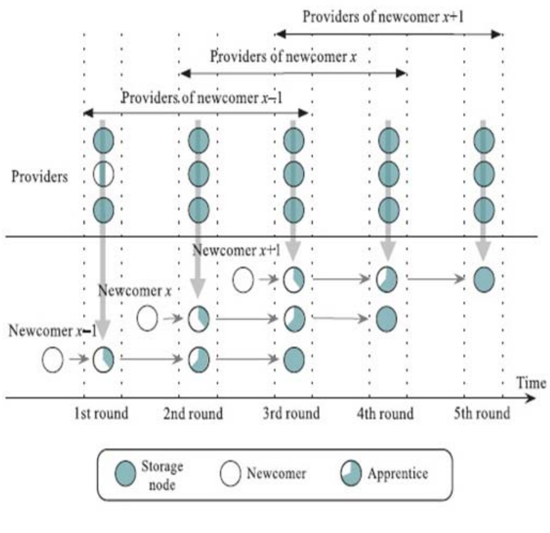


Fig.6 A sample of pipelined repairs with five continuous newcomers.

IV. Saving The Disk I/O Overhead: Locally Repairable Codes

Regenerating codes minimize the repair overhead in the axis of bandwidth utilization. Then again, the minimization of bandwidth utilization does not as a matter of course minimize the amount of information read by suppliers in the repair. Aside from repair-by-transfer regenerating codes, suppliers need to peruse all coded blocks they store and to perform arithmetic operations, with a specific end goal to encode them into the information required by the newcomer.

Like the MDS property that any k coded blocks can recover the first file, regenerating codes support verifiably a property in the data flow chart that a newcomer ought to have the capacity to finish the repair by reaching any k accessible storage nodes as suppliers. Despite the fact that the MDS property goes for information integrity ,this property is not all that important for the cloud storage system, in light of the fact that the MDS property has as of now ensured that k coded blocks can achieve a repair by decoding the first file first and afterward encoding the first file into the specific coded block. On the off chance that this property can be casual, the corresponding erasure coding may be benefited by different properties, e.g., a repair-by-transfer property can be effectively accomplished in Ref. [28].

Hence, in the event that it is conceivable to design erasure coding that any specific coded block can be repaired by some specific coded blocks, the disk I/O overhead in the repair can be significantly lessened, in light of the fact that just a little number of suppliers will be reached by the newcomer. Really, we can watch the pattern of this property from regenerating codes. Since in pipelined repairs suppliers of back to back newcomers must be overlapped, in every repair storage nodes that have as of late seemed should not be chosen as suppliers. Consequently, the number of suppliers can be significantly spared. Aside from regenerating codes, Li et al. [29] observed that random linear codes could likewise be applied in the pipelined repair. In this section, we examine the erasure codes that are all the more explicitly optimized towards this property.

In this paper, we order the erasure coding procedures that accomplish this property as locally repairable codes and present three representative groups of the locally repairable codes: hierarchical codes, self-repairing codes, and simple regenerating codes. At that point we audit the analytical results of the trade-off between the disk I/O overhead and handling against failures.

1. *Hierarchical codes*

While a lost replica must be repaired from the same replica and a lost block coded by MDS codes can be repaired from any k coded blocks, hierarchical codes[30] present another flexible trade-off in the middle of replication and MDS codes, which decrease the number of blocks used with a sacrifice of storage overhead.

As the name recommends, hierarchical codes are built hierarchically. Figure7 outlines a sample of the hierarchical development of hierarchical codes. In Fig. 7a, an example of (2,1) hierarchical codes are developed which creates two efficient blocks and one parity block. Given F_1 and F_2 as first blocks, B_1, B_2 and B_3 are linear mixes of their immediate neighbours, where just B_3, of degree 2, is the parity block. Any two of B_1;B_2; and B_3 have two edge-disjoint paths to F_1 and F_2, recommending that any two of them can repair the other one.

In Fig. 7b, (4,3) hierarchical codes are built from (2,1)c hierarchical odes, repeating the structure of (2,1) hierarchical codes twice and embeddings one more block B_7 joined with every single unique blocks. Clearly, to repair B_7, the greater part of the four unique blocks are required. On the other hand, with respect to the coded blocks in the first structures of (2,1) hierarchical codes, their repair degree, i.e., the number of blocks required to repair a specific coded block, stays to be two.

In the route portrayed over, an occurrence of large hierarchical codes can be built regulated from examples of smaller hierarchical codes. The repair degrees of coded blocks shift from 2 to k. Duminuco and Biersack [30] demonstrated that to repair one coded block, most coded blocks can be repaired by getting to just two coded blocks. Despite the fact that the most worst case stays to be k, just a little proportion of coded blocks fits in with this case. Strictly when countless blocks have been lost, the most worst case turns out to be significantly detectable.

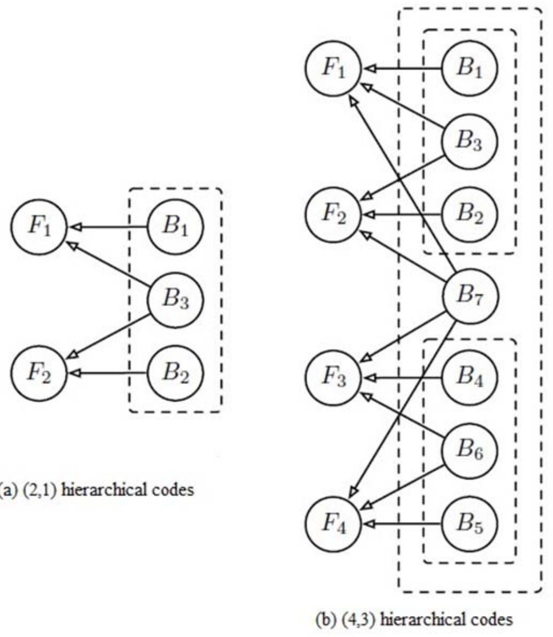


Fig.7 The hierarchical structure of hierarchical codes. This example is originally shown in Ref. [30].

By applying correct regenerating codes in the base structure of hierarchical codes, Huang et al. [31] proposed a group of ER-Hierarchical codes, which consolidate the upsides of hierarchical codes and regenerating codes, such that both a little bandwidth utilization and a low repair degree can be accomplished.

Hierarchical codes offer a low repair degree all things considered. Nonetheless, the MDS property cannot be kept up. Given an example of (k, h) hierarchical codes, not all gatherings of h failures can be handled. Far more atrocious, since the development of hierarchical codes relies on upon the hierarchical structure of the specific example, the capacity of handling failures cannot be anticipated simply taking into account these two parameters. Despite the fact that the structure of the hierarchical codes is given, the capacity of handling failures still cannot be portrayed by explicit equations.

2. *Self-repairing codes*

Not quite the same as hierarchical codes in which the repair level of a coded block may shift from 2 to k, self repairing codes can accomplish a steady repair degree, free of any specific missing block. In addition, depending upon what number of coded blocks are missing, the repair level of a regular coded block can be as low as 2 or 3.

Oggier and Datta [32] proposed *Homomorphic Self Repairing* Codes (HSRC), the first development of self repairing codes, taking into account linear polynomials. Review that Reed-Solomon codes, an ordinary group of MDS codes, is defined by polynomial

$$p(X) = \sum_{i-1}^{k} o_i \ X^{i-1} \qquad (6)$$

over a finite field. A linearly polynomial is defined

$$p(X) = \sum_{i=1}^{k-1} p_i \, X^{q^i} \qquad (7)$$

over a finite field of size 2^m, such that $p(ua + vb) = up(a) + vp(b)$. Thusly, any coded block can be displayed as a linear mixes of a few sets of no less than two other coded blocks. This property does not just give each coded block a low repair degree, yet empowers parallel repairs of various missing blocks also, as newcomers can have diverse options of suppliers to keep away collisions.

Aside from developing linearly polynomials, another group of self-repairing codes exists, which can be manufactured utilizing projective geometry [33]. *Projective geometry Self-Repairing* Codes (PSRC) hold the properties of homomorphic self-repairing codes, furthermore can be built as efficient codes, significantly simplifying decoding methods.

Since self-repairing codes are built over a finite field of size 2^m, the encoding and decoding operations should be possible by XOR operations. In this manner, they accomplish a high computational efficiency. Then again, self-repairing codes cannot keep up the MDS property, unless $k = 2$ for PSRC. On the other hand, the strength likelihood of self-repairing codes is near that of MDS codes.

3. Simple regenerating codes

In spite of the fact that both self-repairing codes and hierarchical codes can accomplish a low repair degree, their flexibilities to the failures of storage nodes are probabilistic, i.e., there is no ensure that a sure number of coded blocks can recover the first file. Subsequently, the storage system must test the coefficients of coded blocks before really getting to the relating suppliers. By and by, the storage system have the capacity to cease themselves from this operation with basic regenerating codes[34], while as yet keeping up a low repair degree and accomplishing the correct repair.

An occurrence of (*n*, *k*, *d*)simple regenerating codes applying so as to recover codes can be developed XOR operations over MDS coded blocks. We take the development of (4, 2, 2)simple regenerating codes as an illustration in Fig. 8. The first file is separated into four fragments, and 8 coded fragments (X_i, y_i, i = 1,...,4) are generated by two examples of(4,2)MDS codes. Node i stores x_i and $y_{(i+1)\,mod\,4}$, as well as the XOR of $x_{(i+2)\,mod\,4}$ and $y_{(i+2)\,mod\,4}$. Along these lines, any fragment can be repaired by getting to two different fragments in Node ((*i*-1) mod 4) and Node ((*i*+1) mod 4)

However "regenerating codes" are incorporated into the name, simple regenerating codes don't accomplish the cut-set bound of regenerating codes and in this way cannot be classified as regenerating codes. Then again, in (*n*, *k*, *d*)simple regenerating codes, every node can be repaired by accessing *d* +1 specific nodes, where *d* can be less than *k* rather than no not as much as *k* in regenerating codes. The newcomer repairs every fragments by XORing comparing two fragments acquired from two different nodes, and in this way the repair is correct. This disk I/O overhead and the bandwidth utilization is $\frac{M}{k} \cdot \frac{2(d+1)}{d}$.

It is anything but not difficult to find out that any two nodes in Fig.8b can recover the first file. Nonetheless, every node ought to store coded fragments size of $\frac{1}{k}$ of the first file, in addition to one more parity fragment. In this manner, (*n*, *k*, *d*) simple regenerating codes bring about extra storage overhead by $\frac{M}{k} \cdot \frac{n}{d}$. Compared with self repairing codes and hierarchal codes , the resilience against failures and storage overhead of simple regenerating codes gets to be predictable.

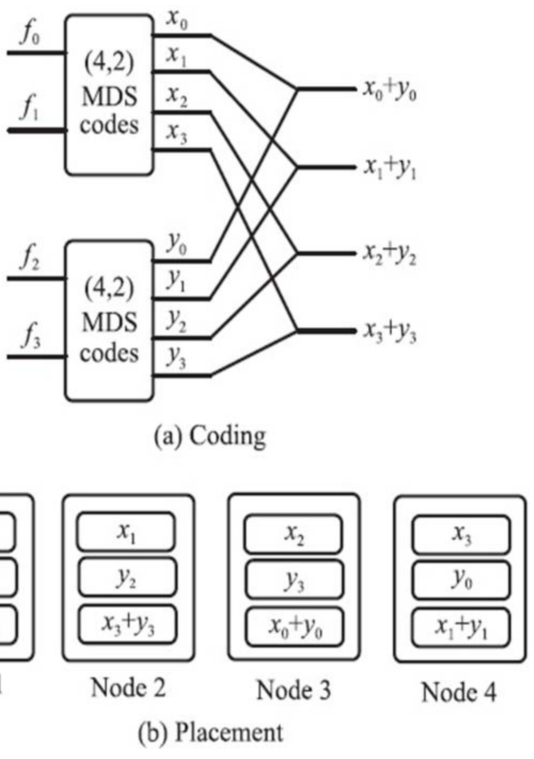


Fig. 8 An example Ref. [34] of (4,2,2) basic recovering codes.

4. Trade-off between the failure tolerance and the repair degree

Locally repairable codes guarantee an exceptionally encouraging property of a low repair degree, suggesting low disk I/O overhead in the repair. Microsoft has their own particular occasion of locally repairable codes which are deployed in its cloud storage system, Windows Azure Storage [5]. Then again, it has been demonstrated that none of the locally repairable codes can safeguard the MDS property. There was no essential comprehension of the failure handling and the repair degree, until Gopalan et al. [35] found a tight bound of the repair degree *d* in wording *n*, *k*, and the minimum distance of the codes *D*. Naturally, the first file can be recovered from any *n* - *D* + 1 coded blocks. Hence, the distance of MDS codes is clearly *n*- *k* + 1. Gopalan et al. [35] demonstrated that in any (*n*, *k*, *d*)codes with the distance *D* which is minimum,

$$n\text{-}k \geq \left\lceil \frac{k}{d} \right\rceil + D - 2 \qquad (8)$$

Given this trade-off, it is anything but not difficult to find out that the repair level of MDS codes cannot be not as much as less than *k*.

Utilizing the bound above, Sathiamoorthy et al. [2] pointed out that there existed (*n*, *k*, *d*) locally repairable codes with the logarithmic repair degree *r* = log *k*, and the minimum distance

$$D = n - (1 + \sigma_k)k + 1, \text{ where } \sigma_k = \frac{1}{r} - \frac{1}{k}. \quad (9)$$

Explicitly, they proposed(16,6,5)locally repairable codes, which have been executed in HDFS-Xorbas, an open source module that keeps running above HDFS (Hadoop File System). Facebook has begun a transitioning deployment of HDFS RAID, an HDFS module that executes (10,4)Reed-Solomon codes. As appeared in Fig. 9, the (16, 6, 5)locally repairable codes are built on the (10, 4)efficient Reed-Solomon codes, including two extra neighbourhood parity blocks.

Accordingly, HDFS-Xorbas is good with the current HDFS RAID, and the system can be moved from HDFS RAID to HDFS-Xorbas by essentially including the two neighbourhood parity blocks. The two nearby parity blocks are linear combinations of the first and last five efficient blocks, individually. Moreover, the coefficients are developed such that the two neighbourhood parity block and the four Reed-Solomon (non-efficient) parity blocks are linearly dependent, i.e., S1 + S2 + S3 = 0. Therefore, any one block can be displayed to as a component of five different blocks, and in this manner can be repaired by five suppliers just. Tests demonstrate that compared with HDFS RAID, approximately half disk I/O and network traffic in the repair have been spared, separately. It is likewise demonstrated that the distance of the (16,10,5)locally repairable codes is 5, accomplishing the bound of the minimum distance with the present repair degree. Papailiopoulos and Dimakis [36] examined the trade-off between repair degree, the minimum distance, and the size of coded blocks α. They demonstrated that the minimum distance is limited as

$$D \leq n - \left\lceil \frac{M}{\alpha} \right\rceil - \left\lceil \frac{M}{d\alpha} \right\rceil + 2. \quad (10)$$

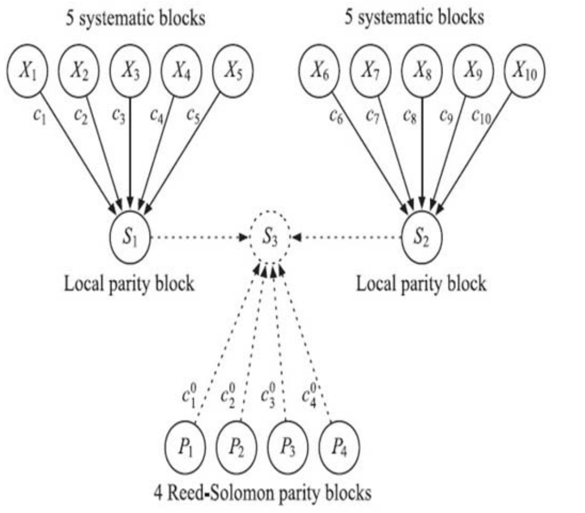


Fig. 9 The (16, 6, 5) locally repairable codes that are developed in Ref. [2] and executed in HDFS-Xorbas. S3 will be repaired by S1 AND S2.

An intriguing viewpoint in this bound is that we can all the while keep up the MDS property and accomplish a arbitrarily low repair degree. Each case of simple regenerating codes is a sample of (n, k, d)locally repairable codes where

$$\alpha = (1 + \frac{1}{d}) \frac{M}{k} \quad (11)$$

V. Concluding Remarks

All through this paper, we give a diagram of the development of coding procedures for cloud storage systems. The way of commodity hardware in the cloud and the huge number of storage devices convey challenges to the outline of cloud storage systems. By introducing erasure coding from regenerating codes to locally repairable codes, we have seen a pattern in the examination of erasure codes for cloud storage, that the configuration objective step by step exchanges from information integrity to resource overhead, and from the bandwidth resource to some other scarcer resource for the cloud storage system, for example, calculation and disk I/O overhead. To spare computational resources, the development of correct regenerating codes has first been considered. The correct repair serves to keep up the efficient erasure codes in the storage system, such that no decoding operations are required to recover the first file. Additionally, the repair-by-transfer regenerating codes help to accomplish a repair process without math operations on both the newcomer and supplier.

To spare disk I/O overhead, locally repairable codes are proposed such that meeting a little number of disks ought to be sufficient to perform a repair process. What's more, some locally repairable codes, for example, simple regenerating codes, bolster the look-up repair that the lost information can be created from a specific block of information put away on some specific disks. The repair-by transfer property is considerably more grounded in light of the fact that just the same information to be repaired will be gotten to from different disks.

Indeed, even with the late advances, there are still some open issues to be researched. In the setting of regenerating codes, there are still a few decisions of parameters with which the presence and the development of regenerating codes are obscure in this way. Furthermore, regenerating codes for correct pipelined repair likewise remains an open issue. As for locally repairable codes, the trade-off between the repair degree and storage overhead has not been built up clearly. Plus, there are some other reasonable considerations that can be talked about mutually with the coding procedure, for example, geological nature of various data centres in the cloud. Given that the cloud storage systems scales all around in numerous data centres, bandwidth, calculation, and the relating geographical heterogeneities should be carefully discussed.

REFERENCES

[1] Brandon Butler. (2013, jan) Gartner: Top 10 cloud storage providers. [Online]. http://www.networkworld.com/article/2162466/cloud-computing/gartner-top-10-cloud-storage-providers.html

[2] A. Loukissas, and A. Vahdat, M. Al-Fares, "A scalable, commodity data center network architecture," in *ACM SIGCOMM 2008* , New York, NY, USA, 2008, pp. 63-74.

[3] P. B. Godfrey, M. J. W. Y. Wu, and A. G. Dimakis, "Network coding for distributed storage," *IEEE Trans. Inform. Theory*, vol. 56, no. 9, pp. 4539 - 4551, sept 2010.

[4] I. Reed and G. Solomon, "Polynomial codes over certain," *Journal of the Society for Industrial and Applied Mathematics,* vol. 8, no. 2, pp. 300-304, 1960.

[5] Huseyin Simitci, Yikang Xu, Aaron Ogus, Brad Calder, Parikshit Gopalan, Jin Li, and Sergey Yekhanin, Cheng Huang, "Erasure coding in Windows Azure Storage," in *USENIX Annual Technical*, boston,MA, 2012.

[6] R. W. Yeung, and N. Cai S.-Y. R. Li, "Linear network coding," *IEEE Trans. on Inform. Theory*, vol. 49, no. 2, pp. 371-381, 2003.

[7] R. Koetter, M. Medard, D. Karger, and M. Effros, T. Ho, "The benefits of coding over routing in a randomized setting," in *IEEE International Symp. Inform. Theory*, 2003, pp. 442-442.

[8] A. Duminuco and E. Biersack, "A practical study of regenerating codes for peer-to-peer backup systems," in *Distributed Computing Systems, 2009. ICDCS '09. 29th IEEE International Conference* , Montreal, QC, 2009, pp. 376 - 384.

[9] N. Shah, and P. Kumar, K. Rashmi, "Optimal exactre generating codes for distributed storage at the MSR and MBR points via a product-matrix construction," *IEEE*, vol. 57, no. 8, pp. 5227-5239, 2011.

[10] C. Suh and K. Ramchandran, "Exact-repair MDS code construction using interference alignment," *IEEE Trans. on Inform. Theory,* vol. 57, no. 3, pp. 1425-1442, 2011.

[11] K. V. Rashmi, P. V. Kumar, N. B. Shah, "Interference alignment in regenerating codes for distributed storage: Necessity and code constructions," *IEEE Trans. on Inform. Theory*, vol. 58, no. 4, pp. 2134-2158, 2012.

[12] S. A. Jafar, and H. Maleki, V. R. Cadambe. (2010) Distributed data storage with minimum storage regenerating codes - Exact and functional repair are asymptotically equally efficient. [Online]. http://arxiv.org/abs/1004.4299

[13] C. Suh and K. Ramchandran. (2010) On the existence of optimal exact-repair MDS codes for distributed storage. [Online]. http://arxiv.org/abs/1004.4663

[14] D. S. Papailiopoulos and A. G. Dimakis, "Distributed storage codes through Hadamard designs," in *Information Theory Proceedings (ISIT), 2011 IEEE International Symposium*, St. Petersburg, 2011, pp. 1230 - 1234.

[15] Y. Wu, "A construction of systematic MDS codes with minimum repair bandwidth," *IEEE Trans. on Inform. Theory*, vol. 57, no. 6, pp. 3738-3741, 2011.

[16] Z. Wang, and J. Bruck, I. Tamo, "MDS array codes with optimal rebuilding," in *IEEE Symposium on Information Theory (ISIT*, St. Petersburg, 2011, pp. 1240-1244.

[17] C. Huang, S. A. Jafar, and J. Li V. R. Cadambe. (2011) repair of MDS codes in distributed storage via subspace interference alignment. [Online]. http://arxiv.org/abs/1106.1250

[18] K. Tati, Y.-C. Cheng, S. Savage, and G. M. R. Bhagwan, "Total recall: System support for automated availability management," in *USENIX Symposium on Networked Systems Design and Implementation (NSDI),* 2004.

[19] Y. Xu, X. Wang, C. Zhan, and P. Li, Y. Hu, "cooperative trecovery of distributed storage systems from multiple losses with network coding," *IEEE Journal on Selected Areas in Communications*, vol. 28, no. 2, pp. 268-276, 2010.

[20] K. Shum, "Cooperative Regenerating Codes for Distributed Storage Systems," in *Communications (ICC), 2011 IEEE International Conference on*, Kyoto, 2011, pp. 1-5.

[21] A. Kermarrec and N. Le, "Repairing multiple failures with coordinated and adaptive regenerating codes," in *IEEE International Symposium on Network Coding (NetCod),* Beijing, 2011, pp. 1-6.

[22] K. Shum and Y. Hu, "Exact minimum-repair-bandwidth cooperative regenerating codes for distributed storage systems," in *Information Theory Proceedings (ISIT), 2011 IEEE International Symposium on*, St. Petersburg, 2011, pp. 1442 - 1446.

[23] A. Wang and Z. Zhang. (2012) Exact cooperative regenerating codes with minimum-repair-bandwidth for distributed storage. [Online]. http://arxiv.org/abs/1207.0879

[24] N. Le Scouarnec, "Exact scalar minimum storage coordinated regenerating codes," in *Information Theory Proceedings (ISIT), 2012 IEEE International Symposium on*, Cambridge, MA, 2012, pp. 1197 - 1201.

[25] K. Shum and Y. Hu, "Functional-repair-by-transfer regenerating codes," in *Information Theory Proceedings (ISIT), 2012 IEEE International Symposium on*, Cambridge, MA, 2012, pp. 1192 - 1196.

[26] P. P. C. Lee, and K. W. Shum, Y. Hu. (2012) Analysis and construction of functional regenerating codes with uncoded repair for distributed storage systems. [Online]. http://arxiv.org/abs/1208.2787v1

[27] K. V. Rashmi, P. V. Kumar, N. B. Shah, "Distributed storage codes with repair-by-transfer and nonachievability of interior points on the storage-bandwidth tradeoff," *IEEE Trans. on Inform. Theory*, vol. 58, no. 3, pp. 1837-1852, 2012.

[28] N. Shah, and P. Kumar, K. Rashmi, "Explicit construction of optimal exact regenerating codes for distributed storage," in *Communication, Control, and Computing, 2009. Allerton 2009. 47th Annual Allerton Conference on*, Monticello, IL, 2009, pp. 1243 - 1249

[29] X. Wang, and B. Li, J. Li, "Cooperative pipelined regeneration in distributed storage systems," in *INFOCOM, 2013 Proceedings IEEE*, Turin, 2013, pp. 2346 - 2354.

[30] 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," in *Peer-to-Peer Computing , 2008. P2P '08. Eighth International Conference on*, Aachen, 2008, pp. 89-98.

[31] E. Biersack, and Y. Peng, Z. Huang, "Reducing Repair Traffic in P2P Backup Systems: Exact Regenerating Codes on Hierarchical Codes," *ACM Transactions on Storage (TOS),* vol. 7, no. 3, p. 10, 2011.

[32] F. Oggier and A. Datta, "Self-repairing homomorphic codes for distributed storage systems," in *INFOCOM, 2011 Proceedings IEEE*, shanghai, 2011, pp. 1215 - 1223.

[33] F. Oggier and A. Datta, "Self-repairing codes for distributed storage—A projective geometric construction," in *Information Theory Workshop (ITW), 2011 IEEE*, Paraty, 2011, pp. 30-34.

[34] Jianqiang Luo, A.G. Dimakis, and Papailiopoulos, D.S Cheng Huang, "Simple regenerating codes: Network coding for cloud storage," in *INFOCOM, 2012 Proceedings IEEE*, Orlando, FL, 2012, pp. 2801 - 2805.

[35] C. Huang, H. Simitci, and S. Yekhanin, P. Gopalan, "On the Locality of Codeword Symbols," *Information Theory, IEEE Transactions on (Volume:58 , Issue: 11)*, vol. 58, no. 11, pp. 6925 - 6934, aug 2012.

[36] D. S. Papailiopoulos and A. G. Dimakis, "Locally Repairable Codes," *Information Theory, IEEE Transactions on*, vol. 60, no. 10, pp. 5843 - 5855, may 2014.