TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 06/11 pp259-272
Volume 18, Number 3, June 2013

Erasure Coding for Cloud Storage Systems: A Survey

Jun Li and Baochun Li*

Abstract: In the current era of cloud computing, data stored in the cloud is being generated at a tremendous speed, and thus the cloud storage system has become one of the key components in cloud computing. By storing a substantial amount of data in commodity disks inside the data center that hosts the cloud, the cloud storage system must consider one question very carefully: how do we store data reliably with a high efficiency in terms of both storage overhead and data integrity? Though it is easy to store replicated data to tolerate a certain amount of data losses, it suffers from a very low storage efficiency. Conventional erasure coding techniques, such as Reed-Solomon codes, are able to achieve a much lower storage cost with the same level of tolerance against disk failures. However, it incurs much higher repair costs, not to mention an even higher access latency. In this sense, designing new coding techniques for cloud storage systems has gained a significant amount of attention in both academia and the industry. In this paper, we examine the existing results of coding techniques for cloud storage systems. Specifically, we present these coding techniques into two categories: regenerating codes and locally repairable codes. These two kinds of codes meet the requirements of cloud storage along two different axes: optimizing bandwidth and I/O overhead. We present an overview of recent advances in these two categories of coding techniques. Moreover, we introduce the main ideas of some specific coding techniques at a high level, and discuss their motivations and performance.

Key words: erasure coding; cloud storage; regenerating codes; locally repairable codes

1 Introduction

With the decreasing of bandwidth and storage pricing, a trend has been observed that leading IT companies, such as Google, Microsoft, and Amazon, build their services inside data centers and provide services globally through a high-bandwidth network. This new paradigm of providing computing services is called *cloud computing*.

In the context of cloud computing, storage has not only been an important component of large-scale cloud services, but also been provided as a virtual storage infrastructure in a pay-as-you-go manner, such as Amazon S3[1]. Moreover, the volume of data stored inside data centers has been observed to be growing even faster than Moore's Law[2,3]. It has been reported that the storage space used for photo storage only in Facebook has been over 20 PB in 2011 and is increasing by 60 TB every week[4].

To meet the requirements of the massive volume of storage, the cloud storage system has to scale out, i.e., storing data in a large number of commodity disks. In this sense, it becomes a major challenge for cloud storage systems to maintain data integrity, due to both the large number of disks and their commodity nature. Even though the number of disk failures is a small portion inside the data centers, there can still be a large number of such failures everyday due to the large number of disks. For example[5], in a Facebook cluster with 3000 nodes, there are typically at least 20 repairs triggered everyday. Apart from storage devices, the

• Jun Li and Baochun Li are with the Department of Electrical and Computing Engineering, University of Toronto, Toronto M5S 3G8, Canada. E-mail: {junli, bli}@eecg.toronto.edu.
∗ To whom correspondence should be addressed.
Manuscript received: 2013-03-09; accepted: 2013-03-15

other systems in the data center, such as the networking or power systems, may cause outages in the data center[6], making data unavailable or even get lost.

To compensate for the data loss and thus to guarantee the integrity of data stored in the cloud, it is natural to store replicas in multiple disks, such that data losses can be tolerated as long as there is at least one replica available. However, replicas can significantly reduce the storage efficiency. For example, if data are stored with 3-way replication, the effective storage space can be no better than $\frac{1}{3}$ of the total consumed storage space.

If the design objective is to maximize the storage efficiency without sacrificing the ability to tolerate disk failures, the storage system should store data encoded by erasure coding. Before the emergence of cloud computing, erasure coding has long been proposed to detect or correct errors in storage or communication systems. For example, RAID 6 can compensate for at most 2 disk failures by parity coding, whose storage efficiency is at most $1-\frac{2}{n}$ where n is the total number of disks. Reed-Solomon codes can provide even more flexibility such that any number of disk failures under a certain threshold can be tolerated. Wuala[7], an online secure storage service, uses Reed-Solomon codes to ensure data integrity, by encoding data with Reed-Solomon codes after encryption.

However, two main drawbacks conventionally prevent erasure coding from being practical and popular in cloud storage. First, to read or write data, the system needs to encode or decode data, leading to a high access latency and a low access throughput due to the CPU limit. Second, though erasure coding stores data as multiple coded blocks, when one coded block gets lost, the system must access multiple coded blocks that are sufficient to recover all the data. It is estimated that even if half of the data in the Facebook cluster are encoded, the repair traffic will saturate the network links in the cluster[5]. This overhead makes the repair with erasure coding very expensive in terms of both bandwidth and disk I/O overhead. Unfortunately, applications hosted in the cloud are sensitive to the disk I/O performance, and bandwidth is always a limited resource inside the data center since most data centers introduce link oversubscription[8].

To meet the requirements of cloud applications for storage, the design of new coding techniques for cloud storage has attracted a substantial amount of interest in the research community. In this paper, we examine recent advances of these coding techniques in the context of cloud storage. Specifically, we introduce coding techniques that optimize the overhead of data repair in two different axes of design objectives: minimizing the bandwidth consumption and optimizing disk I/O overhead. Due to the high frequency of data repairs and the high repair overhead of conventional erasure coding, optimizations in both of these two axes fall into the considerations of data repair.

In the axis of bandwidth consumption, Dimakis et al.[9] proposed a family of regenerating codes. Based on a model inspired by network coding, the repairs of coded data can be modeled into an information flow graph, with a constraint that maintains the ability to tolerate disk failures. Based on this model, an optimal tradeoff between storage and bandwidth can be derived, such that given the amount of data stored on the disk, we can obtain an optimal lower bound of the amount of data that should be transferred during the repair. In the tradeoff curve, two extreme points attract much more attention than interior points, corresponding to the minimum storage cost and the minimum bandwidth cost, respectively. To achieve these two extreme points, many papers proposed instances of regenerating codes that either construct randomized codes that maintain the tolerance against disk failures with a high probability, or construct deterministic codes using interference alignment. With the help of deterministic coding, lost data can be repaired exactly, suggesting that we can construct systematic codes to achieve an access latency as low as replicas. Moreover, cooperative regenerating codes make it possible to repair from multiple disk failures with an even lower bound of bandwidth consumption, with the help of cooperation among participating servers.

Regenerating codes saves bandwidth by not transferring data that are unnecessary to the particular newcomer. It is required that the data sent out of providers must take the corresponding information (i.e., the entropy) required by the newcomer, in order to maintain the tolerance against disk failures. Consequently, only except some special cases, most instances of regenerating codes need to ask providers to send linear combinations of their data to the newcomer. In other words, regenerating codes can only save bandwidth but not disk I/O. Compared with conventional erasure coding, disk I/O will even probably be increased with regenerating codes. With regenerating codes, the eventual amount of data read

from disks during repair is still going to be significantly more than the amount of data written to the newcomer. The excessive disk I/O operations can significantly affect the overall disk I/O performance in the data center.

In this sense, some families of coding techniques have been proposed to reduce disk I/O overhead during repair. Unlike regenerating codes that save the bandwidth consumption by incurring more disk I/O overhead, all of them feature a smaller number of disks accessed in the repair. This feature is achieved by enforcing that data in one disk can only be repaired by data in some certain disks. This idea will make the tolerance against disk failures be sacrificed, but significantly reduce the number of disks to be visited and thus the number of bits to be read. In this paper, we introduce three representative proposals: hierarchical codes, self-repairing codes, and simple regenerating codes. Besides, we show the fundamental tradeoff between failure tolerance and disk I/O overhead.

2 Erasure Coding and Its Performance Metrics

2.1 Erasure coding in storage systems

In a storage system where data are stored on a large number of commodity disks, it is clear that disk failures can not be considered as just exceptions, but as a rule. Therefore, the storage system has to store redundancy such that when a certain number of disks lose data, data can still be accessible from other disks. For example, the N-way replication, which stores N replicas in N different disks, is able to tolerate at most $N-1$ disk failures. Figure 1a illustrates an example of 3-way replication, where the original data are disseminated into 3 different disks and any one disk is able to repair or access the original data. However, N-way replication can only achieve a storage efficiency of $\frac{1}{N}$ at best. Erasure coding, on the other hand, is able to tolerate the same number of disk failures, yet with a much better storage efficiency. Among erasure coding, Maximum Distance Separable (MDS) codes (e.g., Reed-Solomon codes[10]) achieve the optimal storage efficiency.

Suppose that in the storage system, data are organized in the unit of data object, which may correspond to a file or a fix-sized block in different storage systems. Assume that a data object can be stored onto n disks. Given an arbitrary number of k, where $k < n$, (n, k) MDS codes can guarantee to tolerate at most

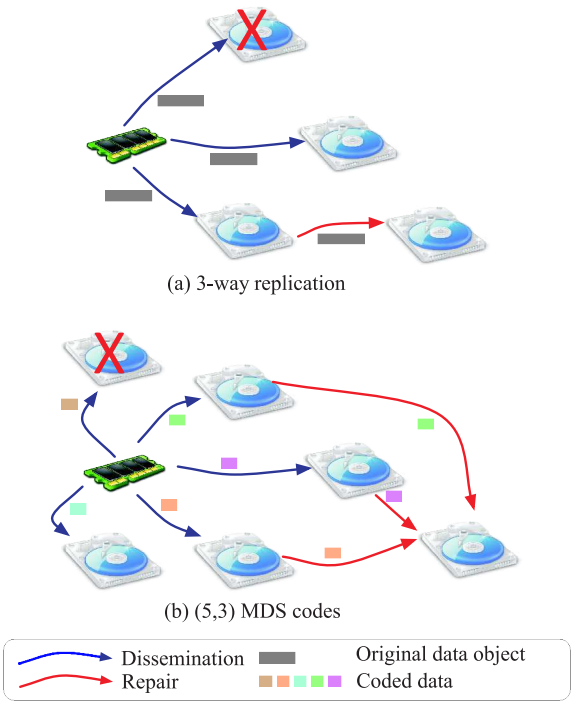

Fig. 1 Comparison of 3-way replication and (5,3) MDS codes. In 3-way replication, the original data object is replicated into 3 different disks, such that any bit of data can be obtained from any one disk. In (5,3) MDS codes, data are encoded into smaller blocks in which any three can be decoded into the original data.

$n-k$ disk failures, i.e., k disks are sufficient to access any bit of the original data. Specifically, the data object is encoded into n coded blocks and are uniformly disseminated into the n disks. Suppose the size of the data object is M bits, the size of each coded block should be $\frac{M}{k}$ bits, if we do not consider the metadata of the data object. In this sense, the storage efficiency of MDS codes is at best $\frac{k}{n}$. Compared to the 3-way replication as shown in Fig. 1b, $(5, 3)$ MDS codes can still tolerate at most 2 disk failures, while improving the storage efficiency by 80%.

To access the data object, the system needs to access k different coded blocks (from k different disks) and recover the original data object by the decoding algorithm of the corresponding MDS codes. Apparently, the decoding algorithm incurs an additional access latency. To improve the access latency, the storage system can use a cache to store one replica of the original data object as well[11].

Recovering the whole data object for data access may sound reasonable in most cases. However, from the point of view of the storage system, it is totally

unnecessary to recover the whole data object if we only need to repair one lost coded block, as what we need is just a very small portion of the data object. Unfortunately, before the appearance of regenerating codes, all MDS codes, to our best knowledge, require to access at least k disks to repair even only one disk, while in the case of replication, to repair one replica we only need to transfer one replica. This requirement can dramatically increase both disk I/O and bandwidth overhead in a data center, and significantly affect the performance of the storage system and other applications hosted in the cloud.

2.2 Performance metrics

It has been made clear that it is not enough to consider only the storage efficiency and the tolerance against disk failures in cloud storage systems. The performance metrics that should be considered when designing erasure coding for cloud storage should also include:

• **Repair bandwidth**. To repair a failed disk, data stored on that disk should be repaired in a replacement disk. The server with the replacement disk, called a *newcomer*, needs to retrieve data from some existing disks. If the servers that host these existing disks, called *providers*, send out the raw data of their coded blocks, the bandwidth used to transmit the existing coded blocks equals the size of these coded blocks and then the newcomer encodes the received data by itself to generate the lost data. However, if encoding operations can be performed both on the newcomer and providers rather than on the newcomer only, a much smaller amount of data can be transferred. As shown in Fig. 2, if data stored in storage nodes are encoded by vector codes, such that each coded block contains more than one coded segments, the bandwidth consumption can be saved when providers send out linear combinations of their coded segments during the repair.

In the pioneering paper of Dimakis et al.[9], a surprising and promising result was shown that the bandwidth used in the repair can be approximately as low as the size of the repaired block by encoding operations of providers and the family of erasure codes that achieves the optimal bandwidth consumption during the repair was called *regenerating codes*.

• **Repair I/O**. Besides bandwidth, another performance metric in the repair for erasure coding is disk I/O at the participating servers. Specifically, the writing operations are performed only at the newcomer, and the amount of data written should equal the size of the coded block. As the writing operations are unavoidable, what we really care is actually the amount of data read from disks of providers.

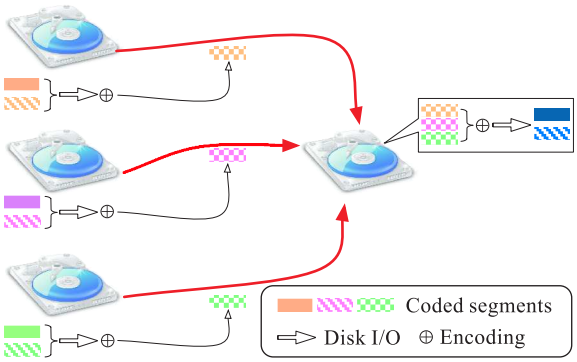

Fig. 2 Suppose that any two storage nodes suffice to recover the original data (k=2), where each provider stores one coded block including two coded segments. With conventional erasure coding, at least two coded blocks must be sent to the newcomer. With data encoded at providers, the amount of data sent to the newcomer can be eliminated into 1.5 coded blocks (3 coded segments), reducing bandwidth consumption by 25%. This figure illustrates the basic idea of regenerating codes.

Similar to the bandwidth consumption, conventional erasure codes will ask for providers to read k blocks in total just to repair one block. As shown in Fig. 2, encoding operations on providers can not reduce the amount of data read, and yet only reduce the amount of data transferred to the newcomer, since the segments sent out of providers are encoded from all coded segments in providers, which all must be read from disks. Therefore, new coding techniques need to be proposed to improve the disk I/O during the repair.

In Fig. 3, we present two examples of techniques that

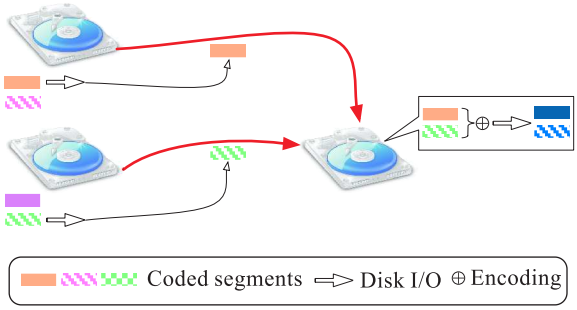

Fig. 3 The disk I/O during the repair can be saved by accessing specific coded segments or/and from specific providers. Suppose that any three storage nodes suffice to recover the original data (k=3), conventional regenerating codes or regenerating codes must read at least 6 coded segments, while in this figure only 2 specific coded segments should be read.

can be used to construct erasure codes with low disk I/O during the repair. One possible way is to obtain specific coded segments from providers, and hence other coded segments in the same provider will not be encoded and thus will not be read. Another technique is to access specific storage nodes as providers, rather than accessing any k storage nodes. Some families of erasure codes have been proposed such that when one storage node fails, the newcomer must access data from a very small number of specific storage nodes.

• **Access latency**. Due to the decoding operations, the access latency of data encoded by erasure coding has to be much larger than replicas. However, systematic codes, in which the original data can be embedded into code blocks, are able to maintain a higher storage efficiency than replicas while achieving a low access latency, since we now only need to access the systematic part with no decoding operations. This attractive property of systematic codes comes with a high price that makes the repair much more complex than non-systematic codes, because the repaired data should be exactly the same as the lost data, at least for the embedded original data. Therefore, the deterministic construction of codes must be considered when designing systematic erasure codes for cloud storage.

• **Storage efficiency**. The storage efficiency denotes the ratio of the amount of the original data to the actual amount of data stored on disks. In other words, given the same amount of data, with a higher storage efficiency we can use a smaller amount of storage space to tolerate the same number of disk failures. MDS codes achieve the optimal storage efficiency, i.e., given n and k, MDS codes can be constructed such that any k among all n coded blocks are sufficient to recover the original data. For example, in Fig. 1, to tolerate two disk failures, $3M$ bits must be stored with replications, while only $\frac{5M}{3}$ bits are required to store with MDS codes. However, if we choose to hold this property, it is unavoidable to incur high disk I/O overhead during the repair[5]. Due to the decreasing costs of large-volume disks, the requirement for storage efficiency can be relaxed such that a much smaller number of disks are required to visit in the repair.

In this paper, we focus on the first two metrics, which are strongly related to the repair in the storage system. Specifically, in Section 3 we present the coding technique, called *regenerating codes*, that is able to achieve the optimal bandwidth bound in the repair. In Section 4, we review the coding techniques along the axis of reducing the disk I/O in the repair. For each coding technique, we discuss their performance in the other two metrics along the way.

3 Tradeoff Between Storage and Bandwidth: Regenerating Codes

3.1 Information flow graph

To investigate the bandwidth consumption in repairs with erasure coding, Dimakis et al.[9] proposed to use the information flow graph, which is a tool used in the analysis of network coding, as a model to characterize the tradeoff between storage and bandwidth.

As shown in Fig. 4, in the information flow graph, all servers can be categorized as the source, storage nodes, and the data collector. The source denotes the server where the data object is originated. Suppose that the size of the data object is M bits. After encoding, coded blocks of α bits are disseminated into n storage nodes. Particularly, the source is represented by a vertex and a storage node is represented by two vertices in the information flow graph. The weight of the edge corresponds to the amount of data stored in the storage node. Thus, after the dissemination, all n storage nodes store α bits and any k of them suffice to recover the original data object, suggesting that $k\alpha \geqslant M$. A virtual vertex called data collector is able to connect to any k storage nodes to recover the original data object.

When a storage node fails, a newcomer does not just connect to k available storage nodes, but d storage nodes as providers ($d \geqslant k$). Different from

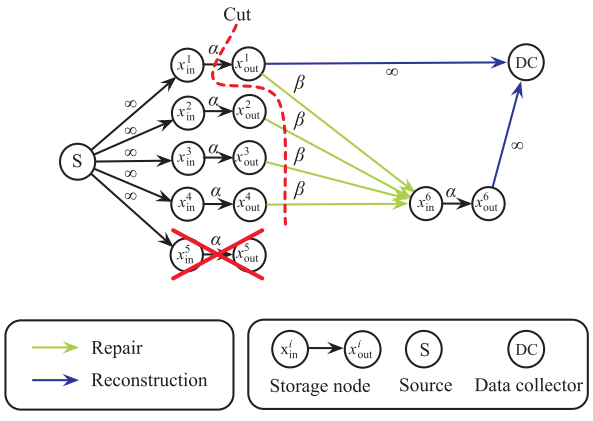

Fig. 4 An illustration of the information flow graph that corresponds to (4,2) MDS codes. This figure was firstly shown in Ref. [9].

conventional MDS codes, the newcomer can receive β bits from each provider, $\beta \leqslant \alpha$, represented by the weight of the edge between the provider and the newcomer. After receiving a total of $r = d\beta$ bits from providers, the newcomer stores α bits and becomes a new storage node. It is required that after any rounds of repairs, the *MDS property* that any k coded blocks suffice to recover the original data object always holds. In other words, the data collector can connect to not only the storage nodes that receive data directly from the source, but the storage nodes repaired from newcomers in repairs as well. Besides, the weights of edges attached to the source or the data collector are all set to be infinity.

Therefore, the repair problem in the storage system can be interpreted as a multicast problem in the information flow graph, where the source is multicasting data to all possible data collectors. It is well known in the multicast problem that the maximum multicast rate equals the minimum cut separating the source from any receivers, and this rate can be achieved through network coding[12]. Thus, it can be proved that the MDS property holds after any rounds of repairs, if the min-cut in the information flow graph is no less that the size of the original data object. By calculating the minimum min-cut in the information flow graph, given d and β we can derive the minimum value of α, and then we obtain the tradeoff curve between α and the total bandwidth consumption r.

3.2 Minimum-storage regenerating codes and minimum-bandwidth regenerating codes

The codes that achieve that the min-cut in the information flow graph equals the amount of the data object is called *regenerating codes*. Given the tradeoff between storage α and bandwidth r, two special cases of regenerating codes interest us most, which correspond to the minimum storage space required at storage nodes and the minimum total bandwidth consumption in the repair, respectively.

The regenerating codes that achieve the minimum storage in storage nodes is called *Minimum-Storage Regenerating (MSR)* codes, where $(\alpha_{\mathrm{MSR}}, r_{\mathrm{MSR}}) = \left(\dfrac{M}{k}, \dfrac{Md}{k(d-k+1)}\right)$. Notice that now α equals to the size of coded blocks of conventional MDS codes, and thus MSR codes can be regarded as MDS codes. However, since $r = \dfrac{Md}{k(d-k+1)} \to \dfrac{M}{k}$ as $d \to \infty$, MSR codes consume bandwidth in the repair approximately the same as the amount of one coded block, while conventional MDS codes consume bandwidth that equals the size of k coded blocks. In this sense, MSR codes save a significant amount of bandwidth in the repair.

The other extreme points in the tradeoff between storage and bandwidth is called *Minimum-Bandwidth Regenerating (MBR)* codes, where $(\alpha_{\mathrm{MBR}}, r_{\mathrm{MBR}}) = \left(\dfrac{2Md}{k(2d-k+1)}, \dfrac{2Md}{k(2d-k+1)}\right)$. MSR codes achieve the minimum bandwidth consumption among regenerating codes. Though MBR codes require more storage than MSR codes, the newcomer only needs to receive exactly the amount of data it needs to repair, and the storage and bandwidth both converge to $\dfrac{M}{k}$ when d is large enough.

To implement regenerating codes, the simplest way (but not necessarily the most efficient way) is to use random linear coding[13] which is inspired by network coding. Dividing the original data object into k blocks, a coded block produced by random linear coding is a 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 corresponds to a byte in the computer, q is usually set to be an integral times of 8 to make encoding operations convenient. Given any k coded blocks and their coding coefficients, the decoding operations are just solving the corresponding linear system with k unknowns (i.e., original blocks) and k equations. When q is large enough, such as 16 or 32, any k coded blocks can be decoded with a very high probability.

On the other hand, the randomized construction of regenerating codes can suffer from a significant computational complexity, especially when parameters are not properly selected[14]. In additional, by no means the randomized regenerating codes can guarantee to repair data exactly as the lost data. Even worse, even a very large Galois field can not ensure that any k code blocks are decodable, but only with a very high probability, due to the randomized coefficients. The number of repairs, however, can not be easily limited, suggesting that after a large number of repairs data integrity can not be maintained, as the randomness accumulates gradually in coded blocks.

Therefore, it is necessary to find explicit construction of regenerating codes, especially for MSR and MBR

codes. Further, it is desirable that the lost block can be repaired exactly given the explicit construction.

3.3 Exact regenerating codes

The most important tool used to construct exact regenerating codes is *interference alignment*, which is initially proposed for wireless communication. The basic idea of interference alignment is that the undesired vectors can be eliminated by aligning them onto the same linear subspace. Figure 5 illustrates how interference alignment helps to achieve exact regenerating codes.

We suppose that in Fig. 5 data are encoded by (n=4, k=2, d=3) MSR codes, i.e., any two of the four nodes can recover the original file. In each storage node, each coded block contains two coded segments, such as (A_1, A_2) in the failed storage node. To recover A_1 and A_2, the newcomer contacts 3 storage nodes as providers and downloads half of a coded block, i.e., a coded segment from each provider to achieve the bandwidth consumption of MSR codes. Notice that each provider owns coded segments containing components of B_1 and B_2, which are undesirable to the newcomer. To eliminate B_1 and B_2, each provider can send a segment in which B_1 and B_2 are aligned onto the same linear subspace of $B_1 + B_2$. Clearly, $B_1 + B_2$ can be eliminated as one unknown and A_1 and A_2 can be decoded by solving three equations with three unknowns.

As for exact MBR codes, Rashmi et al.[15] proposed a Product-Matrix construction which is able to explicitly construct (n, k, d) exact MBR codes on a finite field of size n or higher with any choices of n, k, d if $k \leqslant d < n$. The Product-Matrix construction produces vector MBR codes such that a coded block contains multiple coded segments, just like the example shown in Fig. 5.

As for exact MSR codes, the choices of parameters

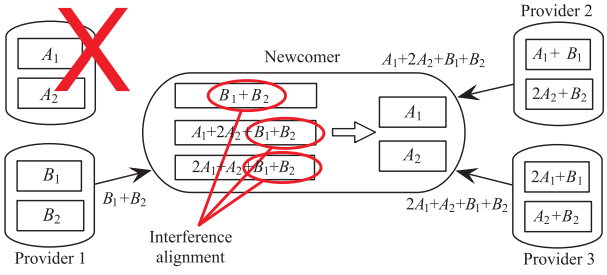

Fig. 5 An example of (4,2,3) exact MSR codes. By interference alignment, the loss of (A_1, A_2) can be repaired exactly since B_1 and B_2 are eliminated by aligning them onto B_1+B_2 in three providers.

become more complicated. Suh and Ramchandran[16] proposed an explicit construction of scalar exact MSR codes where $d \geqslant 2k - 1$, over a finite field of size at least $2(n-k)$. Rashmi et al.[15] improved the choices of parameters such that $d \geqslant 2k - 2$, by constructing exact MSR codes using the Product-Matrix construction, with a larger finite field of size at least $n(d - k + 1)$. In Ref. [17], Shah et al. had proved that no scalar exact MSR codes exist when $d < 2k - 3$.

Cadambe et al.[18] and Suh and Ramchandran[19] showed that any choices of (n, k, d) could be achieved *asymptotically* when constructing vector exact MSR codes, as the field size goes to infinity. Constructing regenerating codes on a very large finite field is impractical due to its overwhelming encoding/decoding overhead. Papailiopoulos and Dimakis[20] showed that $(n = k + 2, k, d = k + 1)$ vector exact MSR codes can be constructed explicitly, combining interference alignment and Hadamard matrices.

Since the exact repair makes much more sense for the systematic part than the parity part of exact regenerating codes, we can construct hybrid regenerating codes that only support the exact repair of the systematic part, while the parity part is still repaired by the functional repair. Wu[21] constructed the $(n, k, d = k + 1)$ hybrid vector MSR codes when $2k \leqslant n$, where the field size is greater than $2\begin{pmatrix} 2n-1 \\ 2k-1 \end{pmatrix}$. Tamo et al.[22] and Cadambe et al.[23] both proposed $(n, k, d = n - 1)$ hybrid MSR codes for arbitrary n and k.

3.4 Cooperative regenerating codes

In some storage systems, such as Total Recall[24], in order to prevent unnecessary repairs incurred by temporary node departures, the system repairs failed nodes in batch when the number of failed nodes reaches a certain threshold. Hu et al.[25] first found that if newcomers can cooperate, there exist *cooperative regenerating codes* that achieve a better tradeoff curve between storage and bandwidth. Still analyzing the min-cut in the information flow graph, Hu et al.[25] showed that $(n, k, d = n - t, t)$ randomized Minimum Storage Cooperative Regenerating (MSCR) codes can achieve the bandwidth consumption of $\frac{M}{k} \cdot \frac{n-1}{n-k}$ with t providers $(n - t \geqslant k)$. Notice that the bandwidth consumption is independent of the number of providers.

A more general result of the bound of bandwidth consumption is shown

that per newcomer, $(\alpha_{\text{MSCR}}, \beta_{\text{MSCR}}) = \left(\frac{M}{k}, \frac{M}{k} \cdot \frac{d+t-1}{d+t-k}\right)$[26,27] and $(\alpha_{\text{MBCR}}, \beta_{\text{MBCR}}) = \left(\frac{M}{k} \cdot \frac{2d+t-1}{2d+t-k}, \frac{M}{k} \cdot \frac{2d+t-1}{2d+t-k}\right)$[27].

Shum and Hu[28] proposed an explicit construction of $(n, k, d = k, t = n-k)$ exact MBCR codes. Wang and Zhang[29] showed that as for all possible values of (n, k, d, t), there exist explicit constructions of exact MBCR on a field of size at least n. On the other hand, when $d = k \neq n-t$, (n, k, d, t) scalar MSCR codes can be constructed[26]. Le Scouarnec[30] discussed the construction of exact MSCR codes with some other choices of parameters when $d \geqslant k = 2$. The existence of exact MSCR codes with other values of parameters still remains an open problem.

3.5 Repair-by-transfer regenerating codes

The regenerating codes we have mentioned above require providers to encode their data to align vectors in a specific linear subspace and the newcomer to encode received data to eliminate undesired components. The arithmetic operations performed on a finite field can be expensive, if the field size is large. Thus, the cloud storage systems will favor the *repair-by transfer* property that in the repair no arithmetic operations are required at providers. With the repair-by-transfer property, the disk I/O overhead can be minimal since only data needed by the newcomer will be read from providers. Moreover, the storage node then can have no intelligence, such that its functionality can be implemented by hardware with a low cost.

Some choices of parameters have been considered to construct the corresponding repair-by-transfer regenerating codes. Shum and Hu[31] and Hu et al.[32] constructed $(n, k = 2, d = n-1)$ and $(n, k = n-2, d = n-1)$ functional repair-by-transfer MSR codes, respectively. On the other hand, $(n, k = n-2, d = n-1)$ exact MBR codes can be constructed over a finite field of size 2[33]. The existence of exact regenerating codes remains an open problem as for most other choices of parameters. The only known result is that if $d \geqslant 2$ and $t \geqslant 2$, any (n, k, d, t) linear exact MBCR codes can not achieve the repair-by-transfer property[29].

Nevertheless, Rashmi et al.[34] proposed an intuitive graph-based construction of repair-by-transfer exact MBR codes, where any missing block can only be repaired from its direct neighbors in the graph. Fractional repetition codes are employed in the construction such that direct neighbors share the same segment and the newcomer only needs to receive replicas of the segments from its neighbors to repair itself, where no arithmetic operations are required not only at providers, but at the newcomer as well. It is shown that any $(n, k, d = n-1)$ MBR codes can be constructed uniquely with this method, $n > k$.

Though repair-by-transfer regenerating codes are able to achieve the minimum disk I/O overhead, only some specific choices of parameters have instances of corresponding regenerating codes so far. Nevertheless, cloud storage systems can have a wide spectrum of parameter choices due to their own requirements. Therefore, it is desirable that erasure coding can achieve low disk I/O overhead with a wide range of parameter choices, even if some properties, such as the MDS property or the storage efficiency, will be sacrificed.

3.6 Regenerating codes for pipelined repair

Except for some specific choices of parameters that have corresponding instances of repair-by-transfer regenerating codes, most instances of regenerating codes require providers to send out linear combinations of their coded blocks to the newcomer. In other words, providers need to read all of their data even though the amount of data to be sent out covers only a very small portion. In this sense, even though the bandwidth consumption can be reduced to the optimum by regenerating codes, the disk I/O increases with the number of providers. As the number of providers can be much larger than k, the disk I/O overhead with regenerating codes will not be improved, but become much severer than MDS codes.

Apparently, as the disk I/O overhead is dependent with the number of providers during the repair, the disk I/O can be saved if the number of providers is reduced while maintaining the other properties of regenerating codes, especially the optimum of the bandwidth consumption. To achieve this goal, Li et al.[35,36] proposed a pipelined repair with minimum-storage regenerating codes. The principle of pipelined repair is that providers of consecutive newcomers are selected to be overlapped, because any k or more than k different providers are sufficient in the repair. Hence, consecutive repairs can be pipelined, as shown in Fig. 6. Still, a newcomer joins when there is one disk failure, but newcomers will not be fully repaired in just one but multiple rounds of repairs and

different newcomers can receive and accumulate data from providers during the repair. After each round of repair, the "eldest" newcomer will contact enough providers and "graduate" to become a storage node.

The most significant benefit of pipelined repair is the reduction of providers in the repair. For example in Fig. 6, the number of providers can be reduced by 67%. In fact, the independent (n,k,d) regenerating codes for the pipelined repair[35] can reduce the number of participating nodes to as few as $2\sqrt{d+1}-1$ in which only $\sqrt{d+1}-1$ are providers, while still maintaining the same bandwidth consumption of (n,k,d) minimum-storage regenerating codes. It is surprising to find that the number of providers can be even less than k, as long as the newcomer is able to receive data from at least d providers before its "graduation". The (n,k,d,t) cooperative pipelined regenerating codes, which can be regarded as a generalization of independent pipelined regenerating codes, have been discussed in Ref. [36], which require $\sqrt{t(d+t)}-t$ providers while maintaining the bandwidth consumption 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 can only be fully repaired after multiple rounds of repairs. The additional storage overhead equals the storage space used by $\sqrt{(d+t)t}-t$ storage nodes. Thus, the storage efficiency will be reduced.

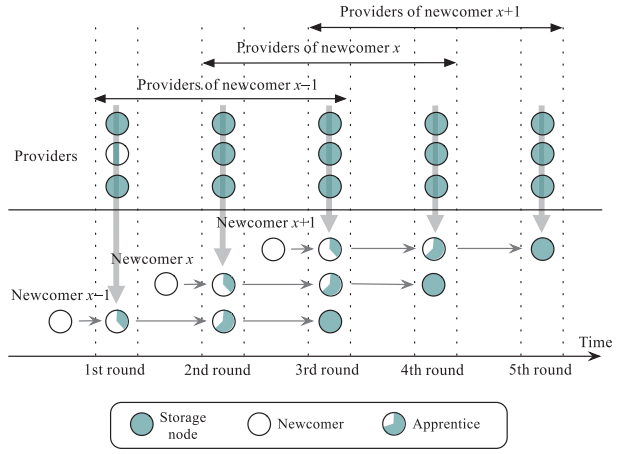

Fig. 6 An example of pipelined repairs with five consecutive newcomers. Suppose that each newcomer must contact at least nine providers to be repaired. When providers are overlapped, only three providers are required in each round of repair while one newcomer can be fully repaired after each round of repair.

However, it can be shown that this overhead is marginal in practical cloud storage systems[36]. Moreover, every block in fully repaired newcomers can still maintain the MDS property, even though the number of providers in the repair can be less than k.

4 Saving the Disk I/O Overhead: Locally Repairable Codes

Regenerating codes optimize the repair overhead in the axis of bandwidth consumption. However, the optimization of bandwidth consumption does not necessarily optimize the amount of data read by providers in the repair. Apart from repair-by-transfer regenerating codes, providers need to read all coded blocks they store and to perform arithmetic operations, in order to encode them into the data required by the newcomer.

Like the MDS property that any k coded blocks can recover the original file, regenerating codes enforce implicitly a property in the information flow graph that a newcomer should be able to finish the repair by contacting any k available storage nodes as providers. Though the MDS property aims for data integrity, this property is not so necessary for the cloud storage system, because the MDS property has already guaranteed that k coded blocks can accomplish a repair by decoding the original file first and then encoding the original file into the particular coded block. If this property can be relaxed, the corresponding erasure coding may be benefited by other properties, e.g., a repair-by-transfer property can be easily achieved in Ref. [34].

Therefore, if it is possible to design erasure coding that any particular coded block can be repaired by some specific coded blocks, the disk I/O overhead in the repair can be significantly reduced, because only a very small number of providers will be contacted by the newcomer. Actually, we can observe the trend of this property from regenerating codes. Since in pipelined repairs providers of consecutive newcomers must be overlapped, in each repair storage nodes that have recently appeared must not be selected as providers. In return, the number of providers can be significantly saved. Apart from regenerating codes, Li et al.[36] found that random linear codes could also be applied in the pipelined repair. In this section, we discuss the erasure codes that are more explicitly optimized towards this property.

In this paper, we categorize the erasure coding techniques that achieve this property as *locally repairable codes* and introduce three representative families of the locally repairable codes: hierarchical codes, self-repairing codes, and simple regenerating codes. Then we review the analytical results of the tradeoff between the tolerance against failures and the disk I/O overhead.

4.1 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[37] introduce a new flexible tradeoff between replication and MDS codes, which reduce the number of blocks accessed with a sacrifice of storage overhead.

As the name suggests, hierarchical codes are constructed in a hierarchical way. Figure 7 illustrates an example of the hierarchical construction of hierarchical codes. In Fig. 7a, an instance of $(2,1)$ hierarchical codes are constructed which produces two systematic blocks and one parity block. Given F_1 and F_2 as original blocks, B_1, B_2, and B_3 are linear combinations of their direct neighbors, where only 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, suggesting that any two of them can repair the other one.

In Fig. 7b, $(4,3)$ hierarchical codes are constructed from $(2,1)$ hierarchical codes, replicating the structure

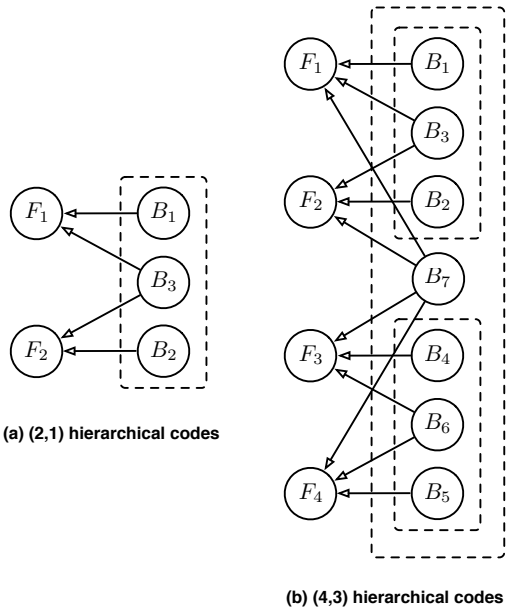

Fig. 7 The hierarchical structure of hierarchical codes. This example is originally shown in Ref. [37].

of $(2,1)$ hierarchical codes twice and inserting one more block B_7 connected to all original blocks. Apparently, to repair B_7, all of the four original blocks are required. However, as for the coded blocks in the original structures of $(2,1)$ hierarchical codes, their repair degree, i.e., the number of blocks required to repair a particular coded block, remains to be two.

In the way described above, an instance of large hierarchical codes can be constructed step by step from instances of smaller hierarchical codes. The repair degrees of coded blocks vary from 2 to k. Duminuco and Biersack[37] showed that to repair one coded block, most coded blocks can be repaired by accessing only two coded blocks. Though the worst case remains to be k, only a very small ratio of coded blocks belongs to this case. Only after a large number of coded blocks have been lost, the worst case becomes significantly noticeable.

By applying exact regenerating codes in the minimum structure of hierarchical codes, Huang et al.[38] proposed a family of ER-Hierarchical codes, which combine the advantages of regenerating codes and hierarchical codes, such that both a small bandwidth consumption and a low repair degree can be achieved.

Hierarchical codes offer a low repair degree on average. However, the MDS property can not be maintained. Given an instance of (k,h) hierarchical codes, not all groups of h failures can be tolerated. Even worse, since the construction of hierarchical codes depends on the hierarchical structure of the specific instance, the ability of failure tolerance can not be predicted just based on these two parameters. Even though the structure of the hierarchical codes is given, the ability of failure tolerance still can not be characterized by explicit formulas.

4.2 Self-repairing codes

Different from hierarchical codes in which the repair degree of a coded block may vary from 2 to k, self-repairing codes can achieve a constant repair degree, independent of any specific missing block. Moreover, depending on how many coded blocks are missing, the repair degree of a typical coded block can be as low as 2 or 3.

Oggier and Datta[39] proposed Homomorphic Self-Repairing Codes (HSRC), the first construction of self-repairing codes, based on linearly polynomials. Recall that Reed-Solomon codes, a typical family of MDS

codes, is defined by polynomial

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

over a finite field. A linearly polynomial is defined as

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

over a finite field of size 2^m, such that $p(ua + vb) = up(a) + vp(b)$. Therefore, any coded block can be represented as a linear combination of some pairs of at least two other coded blocks. This property does not only give each coded block a very low repair degree, but enables parallel repairs of multiple missing blocks as well, as newcomers can have different choices of providers to avoid collisions.

Apart from constructing linearly polynomials, another family of self-repairing codes exists, which can be built using projective geometry[40]. Projective geometry Self-Repairing Codes (PSRC) retain the properties of homomorphic self-repairing codes, and also can be constructed as systematic codes, significantly simplifying the decoding procedure.

Since self-repairing codes are constructed over a finite field of size 2^m, the encoding and decoding operations can be done by XOR operations. Thus, they achieve a high computational efficiency. On the other hand, self-repairing codes can not maintain the MDS property, unless $k = 2$ for PSRC. However, the resilience probability of self-repairing codes is close to that of MDS codes.

4.3 Simple regenerating codes

Though both hierarchical codes and self-repairing codes can achieve a low repair degree, their resiliences to the failures of storage nodes are probabilistic, i.e., there is no guarantee that a certain number of coded blocks can recover the original file. Thus, the storage system must test the coefficients of coded blocks before actually accessing the corresponding providers. Nevertheless, the storage systems are able to refrain themselves from this operation with simple regenerating codes[41], while still maintaining a low repair degree and achieving the exact repair.

An instance of (n, k, d) simple regenerating codes can be constructed by applying XOR operations over MDS coded blocks. We take the construction of $(4, 2, 2)$ simple regenerating codes as an example in Fig. 8. The original file is divided into four segments, and 8 coded segments $(x_i, y_i, i = 1, \cdots, 4)$ are produced by two instances of $(4, 2)$ MDS codes. Node i stores not only x_i and $y_{(i+1) \bmod 4}$, but also the XOR of $x_{(i+2) \bmod 4}$ and $y_{(i+2) \bmod 4}$. Therefore, any segment can be repaired by accessing two other segments in Node $((i - 1) \bmod 4)$ and Node $((i + 1) \bmod 4)$.

Though "regenerating codes" are included in the name, simple regenerating codes do not achieve the cut-set bound of regenerating codes and thus can not be categorized as regenerating codes. On the other hand, in (n, k, d) simple regenerating codes, each node can be repaired by visiting $d + 1$ specific nodes, where d can be fewer than k instead of no less than k in regenerating codes. The newcomer repairs each segments by XORing corresponding two segments obtained from two other nodes, and thus the repair is exact. This disk I/O overhead and the bandwidth consumption is $\frac{M}{k} \cdot \frac{2(d+1)}{d}$.

It is easy to find out that any two nodes in Fig. 8b can recover the original file. However, each node should store coded segments size of $\frac{1}{k}$ of the original file, plus one more parity segment. Therefore, (n, k, d) simple regenerating codes incur additional storage overhead by $\frac{M}{k} \cdot \frac{n}{d}$. Compared with hierarchical codes and self-repairing codes, the tolerance against failures and storage overhead of simple regenerating codes becomes predictable.

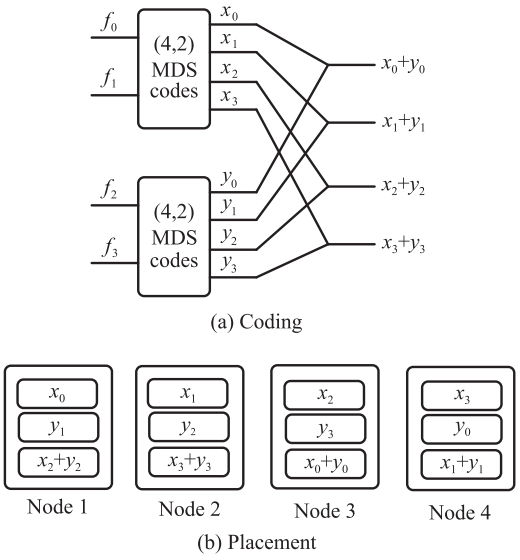

Fig. 8 An example[41] of (4,2,2) simple regenerating codes, where any 2 nodes can recover the original file and one lost node can be repaired from three other nodes.

4.4 Tradeoff between the failure tolerance and the repair degree

Locally repairable codes claim a very promising property of a low repair degree, suggesting low disk I/O overhead in the repair. Microsoft has deployed their own instance of locally repairable codes in its cloud storage system, Windows Azure Storage[11]. However, it has been shown that none of the locally repairable codes can preserve the MDS property. There was no fundamental understanding of the failure tolerance and the repair degree, until Gopalan et al.[42] discovered a tight bound of the repair degree d in terms n, k, and the minimum distance of the codes D. Intuitively, the original file can be recovered from any $n - D + 1$ coded blocks. Thus, the distance of MDS codes is apparently $n - k + 1$. Gopalan et al.[42] proved that in any (n, k, d) codes with the minimum distance D, $n - k \geqslant \left\lceil \frac{k}{d} \right\rceil + D - 2$. Given this tradeoff, it is easy to find out that the repair degree of MDS codes can not be less than k.

Using the bound above, Sathiamoorthy et al.[5] pointed out that there existed (n, k, d) locally repairable codes with the logarithmic repair degree $r = \log k$, and the minimum distance $D = n - (1 + \sigma_k)k + 1$, where $\sigma_k = \frac{1}{r} - \frac{1}{k}$. Explicitly, they proposed $(16, 6, 5)$ locally repairable codes, which have been implemented in HDFS-Xorbas, an open source module that runs above HDFS (Hadoop File System). Facebook has started a transitioning deployment of HDFS RAID, an HDFS module that implements $(10, 4)$ Reed-Solomon codes. As shown in Fig. 9, the $(16, 6, 5)$ locally repairable codes are built on the $(10, 4)$ systematic Reed-Solomon codes, including two additional local parity blocks. Thus, HDFS-Xorbas is compatible with the current HDFS RAID, and the system can be migrated from HDFS RAID to HDFS-Xorbas by simply adding the two local parity blocks. The two local parity blocks are linear combinations of the first and last five systematic blocks, respectively. In addition, the coefficients are constructed such that the two local parity block and the four Reed-Solomon (non-systematic) parity blocks are linearly dependent, i.e., $S_1 + S_2 + S_3 = 0$. Thus, any one block can be represented as a function of five other blocks, and thus can be repaired by five providers only. Experiments show that compared with HDFS RAID, approximately 50% disk I/O and network traffic in the repair have been saved, respectively. It is also proved that the distance of the $(16, 10, 5)$ locally repairable codes is 5, achieving the bound of the minimum distance with the present repair degree.

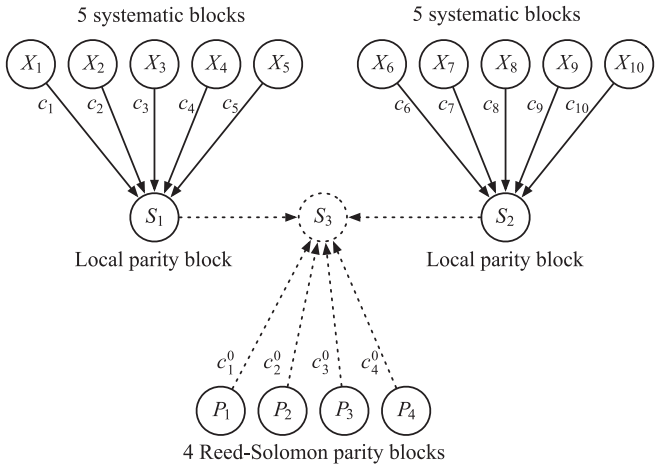

Fig. 9 The (16, 6, 5) locally repairable codes that are constructed in Ref. [5] and implemented in HDFS-Xorbas. S_3 does not exist in the system, but only logically. Since S_1+S_2=S_3, the four Reed-Solomon parity blocks will be repaired by S_1 and S_2.

Papailiopoulos and Dimakis[43] investigated the tradeoff between repair degree, the minimum distance, and the size of coded blocks α. They showed that the minimum distance is bounded as $D \leqslant n - \left\lceil \frac{M}{\alpha} \right\rceil - \left\lceil \frac{M}{d\alpha} \right\rceil + 2$. An interesting perspective in this bound is that we can simultaneously maintain the MDS property and achieve an arbitrarily low repair degree. Every instance of simple regenerating codes is an example of (n, k, d) locally repairable codes where $\alpha = \left(1 + \frac{1}{d}\right)\frac{M}{k}$.

5 Concluding Remarks

Throughout this paper, we give an overview of the advance of coding techniques for cloud storage systems. The nature of commodity hardware in the cloud and the large number of storage devices bring challenges to the design of cloud storage systems. By introducing erasure coding from regenerating codes to locally repairable codes, we have witnessed a trend in the research of erasure codes for cloud storage, that the design objective gradually transfers from data integrity to resource overhead, and from the bandwidth resource to some other scarcer resource for the cloud storage system, such as computation and disk I/O overhead.

To save computational resources, the construction of exact regenerating codes has first been considered. The

exact repair helps to maintain the systematic erasure codes in the storage system, such that no decoding operations are required to recover the original file. Moreover, the repair-by-transfer regenerating codes help to achieve a repair process without arithmetic operations on both the newcomer and providers.

To save disk I/O overhead, locally repairable codes are proposed such that visiting a very small number of disks should be enough to accomplish a repair process. In addition, some locally repairable codes, such as simple regenerating codes, support the look-up repair that the lost data can be produced from a specific part of data stored on some particular disks. The repair-by-transfer property is even stronger because only the same data to be repaired will be accessed from other disks.

Even with the recent advances, there are still some open problems to be investigated. In the context of regenerating codes, there are still some choices of parameters with which the existence and the construction of regenerating codes are unknown so far. In addition, regenerating codes for exact pipelined repair also remains an open problem. With respect to locally repairable codes, the tradeoff between the repair degree and storage overhead has not been established clearly. Besides, there are some other practical considerations that can be discussed jointly with the coding technique, such as geographical nature of multiple data centers in the cloud. Given that the cloud storage system scales globally in multiple data centers, bandwidth, computation, and the corresponding geographical heterogeneities should be carefully discussed.

References

[1] Amazon simple storage service (Amazon S3), http://aws.amazon.com/s3/, 2013.

[2] Worl' data more than doubling every two years — Driving big data opportunity, new IT roles, http://www.emc.com/about/news/press/2011/20110628-01.htm, 2013.

[3] IDC says world's storage is breaking Moore's law, more than doubling every two years, http://enterprise.media.seagate.com/2011/06/inside-it-storage/idc-says-worlds-storage-is-breaking-moores-law-more-than-doubling-every-two-years/, 2012.

[4] D. Beaver, S. Kumar, H. C. Li, J. Sobel, and P. Vajgel, Finding a needle in Haystack: Facebook's photo storage, in *Proc. 9th USENIX Conference on Operating Systems Design and Implementation (OSDI)*, 2010.

[5] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur, XORing elephants: Novel erasure codes for big data, *Proc. VLDB Endowment*, to appear.

[6] A. W. Kosner, Amazon cloud goes down Friday night, taking Netflix, Instagram And Pinterest with it, http://www.forbes.com/sites/anthonykosner/2012/06/30/amazon-cloud-goes-down-friday-night-taking-netflix-instagram-and-pinterest-with-it/, Forbes, 2012, June 30.

[7] Wuala, http://www.wuala.com/en/learn/technology, 2012.

[8] M. Al-Fares, A. Loukissas, and A. Vahdat, A scalable, commodity data center network architecture, in *Proc. ACM SIGCOMM Conference on Data Communication (SIGCOMM)*, 2008.

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

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

[11] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin, Erasure coding in windows azure storage, in *Proc. USENIX Annual Technical Conference (USENIX ATC)*, 2012.

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

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

[14] A. Duminuco and E. Biersack, A practical study of regenerating codes for peer-to-peer backup systems, in *Proc. IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2009, pp. 376-384.

[15] K. Rashmi, N. Shah, and P. Kumar, Optimal exact-regenerating codes for distributed storage at the MSR and MBR points via a product-matrix construction, *IEEE Trans. on Inform. Theory*, vol. 57, no. 8, pp. 5227-5239, 2011.

[16] 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.

[17] N. B. Shah, K. V. Rashmi, P. V. Kumar, and K. Ramchandran, 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.

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

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

[20] D. S. Papailiopoulos and A. G. Dimakis, Distributed storage codes through hadamard designs, in *Proc. IEEE International Symposium on Information Theory (ISIT)*, 2011, pp. 1230-1234.
[21] 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.
[22] I. Tamo, Z. Wang, and J. Bruck, MDS array codes with optimal rebuilding, in *Proc. IEEE International Symposium on Information Theory (ISIT)*, 2011, pp. 1240-1244.
[23] V. R. Cadambe, C. Huang, S. A. Jafar, and J. Li, Optimal repair of MDS codes in distributed storage via subspace interference alignment, http://arxiv.org/abs/1106.1250, 2011.
[24] R. Bhagwan, K. Tati, Y.-C. Cheng, S. Savage, and G. M. Voelker, Total recall: System support for automated availability management, in *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2004.
[25] Y. Hu, Y. Xu, X. Wang, C. Zhan, and P. Li, Cooperative recovery 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.
[26] K. Shum, Cooperative regenerating codes for distributed storage systems, in *IEEE International Conference on Communications (ICC)*, 2011.
[27] A. Kermarrec and N. Le, Repairing multiple failures with coordinated and adaptive regenerating codes, in *IEEE International Symposium on Network Coding (NetCod)*, 2011.
[28] K. Shum and Y. Hu, Exact minimum-repair-bandwidth cooperative regenerating codes for distributed storage systems, in *Proc. IEEE International Symposium on Information Theory (ISIT)*, 2011.
[29] A. Wang and Z. Zhang, Exact cooperative regenerating codes with minimum-repair-bandwidth for distributed storage, http://arxiv.org/abs/1207.0879, 2012.
[30] N. Le Scouarnec, Exact scalar minimum storage coordinated regenerating codes, in *Proc. IEEE International Symposium on Information Theory (ISIT)*, 2012.
[31] K. Shum and Y. Hu, Functional-repair-by-transfer regenerating codes, in *Proc. IEEE International Symposium on Information Theory (ISIT)*, 2012.
[32] Y. Hu, P. P. C. Lee, and K. W. Shum, Analysis and construction of functional regenerating codes with uncoded repair for distributed storage systems, http://arxiv.org/abs/1208.2787v1, 2012.
[33] N. B. Shah, K. V. Rashmi, P. V. Kumar, and K. Ramchandran, 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.
[34] K. Rashmi, N. Shah, and P. Kumar, Explicit construction of optimal exact regenerating codes for distributed storage, in *Forty-Seventh Annual Allerton Conference on Communication, Control, and Computing*, 2009.
[35] J. Li, X. Wang, and B. Li, Pipelined regeneration with regenerating codes for distributed storage systems, in *Proc. IEEE International Symposium on Network Coding (NetCod)*, 2011.
[36] J. Li, X. Wang, and B. Li, Cooperative pipelined regeneration in distributed storage systems, in *Proc. IEEE INFOCOM*, to appear.
[37] A. Duminuco and E. W. Biersack, Hierarchical codes: A flexible trade-off for erasure codes in peer-to-peer storage systems, *Peer-to-Peer Networking and Applications*, vol. 3, no. 1, pp. 52-66, 2010.
[38] Z. Huang, E. Biersack, and Y. Peng, 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.
[39] F. Oggier and A. Datta, Self-repairing homomorphic codes for distributed storage systems, in *Proc. IEEE INFOCOM*, 2011.
[40] F. Oggier and A. Datta, Self-repairing codes for distributed storage—A projective geometric construction, in *IEEE Information Theory Workshop (ITW)*, 2011.
[41] D. S. Papailiopoulos, J. Luo, A. G. Dimakis, C. Huang, and J. Li, Simple regenerating codes: Network coding for cloud storage, in *Proc. IEEE INFOCOM*, 2012, pp. 2801-2805.
[42] P. Gopalan, C. Huang, H. Simitci, and S. Yekhanin, On the locality of codeword symbols, *IEEE Transactions on Information Theory*, vol. 58, no. 11, pp. 6925-6934, 2012.
[43] D. S. Papailiopoulos and A. G. Dimakis, Locally repairable codes, in *Proc. IEEE International Symposium on Information Theory (ISIT)*, 2012, pp. 2771-2775.

Baochun Li received his PhD degree from the Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, in 2000. Since then, he has been with the Department of Electrical and Computer Engineering at the University of Toronto, where he is currently a professor. He holds the Bell Canada Endowed Chair in Computer Engineering since August 2005. His research interests include large-scale distributed systems, cloud computing, peer-to-peer networks, applications of network coding, and wireless networks. He is a member of the ACM and a senior member of the IEEE.

Jun Li received his BS and MS degrees from the School of Computer Science, Fudan Univeristy, China, in 2009 and 2012. He is currently working towards his PhD degree at the Department of Electrical and Computer Engineering, University of Toronto. His research interests include distributed storage and cloud computing.