

Building Reliable Storage Clouds: Models, Fundamental Tradeoffs, and Solutions

Ulaş C. Kozat

Huawei R&D

ulas.kozat@huawei.com

Guanfeng Liang

LinkedIn

gliang@linkedin.com

now

the essence of knowledge

Boston — Delft



Documents

Document Sharing Hub

Foundations and Trends[®] in Networking

Published, sold and distributed by:

now Publishers Inc.
PO Box 1024
Hanover, MA 02339
United States
Tel. +1-781-985-4510
www.nowpublishers.com
sales@nowpublishers.com

Outside North America:

now Publishers Inc.
PO Box 179
2600 AD Delft
The Netherlands
Tel. +31-6-51115274

The preferred citation for this publication is

U. C. Kozat and G. Liang. *Building Reliable Storage Clouds: Models, Fundamental Tradeoffs, and Solutions*. Foundations and Trends[®] in Networking, vol. 9, no. 4, pp. 219–315, 2014.

This Foundations and Trends[®] issue was typeset in L^AT_EX using a class file designed by Neal Parikh. Printed on acid-free paper.

ISBN: 978-1-68083-065-1

© 2015 U. C. Kozat and G. Liang

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, mechanical, photocopying, recording or otherwise, without prior written permission of the publishers.

Photocopying. In the USA: This journal is registered at the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923. Authorization to photocopy items for internal or personal use, or the internal or personal use of specific clients, is granted by now Publishers Inc for users registered with the Copyright Clearance Center (CCC). The 'services' for users can be found on the internet at: www.copyright.com

For those organizations that have been granted a photocopy license, a separate system of payment has been arranged. Authorization does not extend to other kinds of copying, such as that for general distribution, for advertising or promotional purposes, for creating new collective works, or for resale. In the rest of the world: Permission to photocopy must be obtained from the copyright owner. Please apply to now Publishers Inc., PO Box 1024, Hanover, MA 02339, USA; Tel. +1 781 871 0245; www.nowpublishers.com; sales@nowpublishers.com

now Publishers Inc. has an exclusive license to publish this material worldwide. Permission to use this content must be obtained from the copyright license holder. Please apply to now Publishers, PO Box 179, 2600 AD Delft, The Netherlands, www.nowpublishers.com; e-mail: sales@nowpublishers.com

Foundations and Trends® in Networking

Volume 9, Issue 4, 2014

Editorial Board

Editor-in-Chief

Anthony Ephremides
University of Maryland
United States

Editors

François Baccelli
University of Texas, Austin

Victor Bahl
Microsoft Research

Helmut Bölcskei
ETH Zurich

J.J. Garcia-Luna Aceves
UC Santa Cruz

Andrea Goldsmith
Stanford University

Roch Guerin
Washington University in Saint Louis

Bruce Hajek
UIUC

Jean-Pierre Hubaux
EPFL

Frank Kelly
University of Cambridge

P.R. Kumar
Texas A&M University

Steven Low
Caltech

Eytan Modiano
MIT

Keith Ross
Polytechnic Institute of NYU

Henning Schulzrinne
Columbia University

Mani Srivastava
UCLA

Leandros Tassiulas
Yale University

Lang Tong
Cornell University

Ozan Tonguz
Carnegie Mellon University

Don Towsley
University of Massachusetts, Amherst

Nitin Vaidya
UIUC

Pravin Varaiya
UC Berkeley

Roy Yates
Rutgers University

Raymond Yeung
Chinese University of Hong Kong

Editorial Scope

Topics

Foundations and Trends® in Networking publishes survey and tutorial articles in the following topics:

- Modeling and analysis of:
 - Ad hoc wireless networks
 - Sensor networks
 - Optical networks
 - Local area networks
 - Satellite and hybrid networks
 - Cellular networks
 - Internet and web services
- Protocols and cross-layer design
- Network coding
- Energy-efficiency incentives/pricing/utility-based
- Games (co-operative or not)
- Security
- Scalability
- Topology
- Control/Graph-theoretic models
- Dynamics and asymptotic behavior of networks

Information for Librarians

Foundations and Trends® in Networking, 2014, Volume 9, 4 issues. ISSN paper version 1554-057X. ISSN online version 1554-0588. Also available as a combined paper and online subscription.

Foundations and Trends® in Networking
Vol. 9, No. 4 (2014) 219–315
© 2015 U. C. Kozat and G. Liang
DOI: 10.1561/13000000051

now
the essence of knowledge

Building Reliable Storage Clouds: Models, Fundamental Tradeoffs, and Solutions

Ulaş C. Kozat
Huawei R&D
ulas.kozat@huawei.com

Guanfeng Liang
LinkedIn
gliang@linkedin.com

Contents

1	Introduction	2
1.1	Techniques to Improve the Performance of Distributed Storage	5
1.2	Distributed Storage Systems and Operational Characteristics	9
1.3	Book Organization	15
2	Data Durability/Availability, Storage Efficiency, and Network Capacity	16
2.1	MDS Codes	18
2.2	Fountain/Rateless Codes	21
2.3	Regenerating/Repair Codes	24
3	Cloud Storage with high I/O Performance	46
3.1	Models	51
3.2	Black Box Models and Performance Analysis	54
3.3	White Box Models and Performance Analysis	78
3.4	Summary	88
4	Conclusion	89
	References	92

Abstract

Distributed storage has been an active research area for decades. With the proliferation of cloud computing, there has been a rejuvenated interest in two perspectives. The first perspective is seen through the lenses of the cloud providers: how should we build global storage services for cloud hosted services and applications at scale with high reliability and availability guarantees, but also in a cost effective way? The second perspective is seen through the lenses of the service providers that utilize public clouds: how can we achieve high I/O performance over cloud storage within a cost budget? In this manuscript, we first present various kinds of distributed storage systems, their operational characteristics and the key techniques to improve their performance. We then focus on cloud storage, exclusively. Cloud storage has massive scales with the promise to provide as much storage capacity as their tenants demand. Cloud providers also promise very high durability, availability, and I/O performance. In this context, we cover the fundamental tradeoffs between storage efficiency and network bandwidth as well as I/O throughput and latency. Erasure codes play an essential role in these tradeoffs and, thus, we also present their design and usage in the context of cloud storage broadly. We pay particular attention on various queuing models and the corresponding performance analysis in the presence of coded storage. We provide exact and approximate solutions under various settings and assumptions. We describe optimal or near-optimal scheduling and coding strategies that are established based on these analyses.

U. C. Kozat and G. Liang. *Building Reliable Storage Clouds: Models, Fundamental Tradeoffs, and Solutions*. Foundations and Trends[®] in Networking, vol. 9, no. 4, pp. 219–315, 2014.
DOI: 10.1561/13000000051.

1

Introduction

Informally and in the broadest sense, distributed storage systems refer to a system, where the resources of more than one physical storage node are pooled together. The expectations from such systems are many folds: (1) Provide a high aggregate storage capacity, which may not be feasible or too expensive to provision with one node. (2) Avoid single point of failure, e.g., if one storage node fails, the remaining "healthy" nodes should continue to provide access to the information stored on them. (3) Provide high throughput, e.g., utilize multiple pathways to multiple storage nodes to multiplex input/output (I/O) speeds of individual storage nodes. (4) Provide low delay by avoiding/preventing congested nodes.

The landscape of distributed storage ranges from storage arrays directly attached to a server to highly distributed peer to peer (P2P) systems interconnected over the Internet. The major change we have seen over the last decade is that massive data center operators built computing, storage and networking platforms and started leasing them to general public following a utility based pricing. Popular examples include Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform, etc. More and more services are running over public clouds

with a wide range of resource usage. In fact, even massive services such as Netflix and DropBox are built over such cloud services. Large data centers host tens of thousands of physical switches, hundreds of thousands physical servers and virtual switches, and millions of CPU cores. To cope with this scale, cloud technologies continuously evolve on multiple fronts. High performance machine virtualization, data-center centric low-delay high-bandwidth transport layer protocols, new network architectures (e.g., software defined networks - SDN), new management and infrastructure optimization solutions, new inter-connect technologies, and power efficiency are among the most important areas to boost up the performance, efficiency, and manageability of these massive systems.

Cloud storage is offered at different pricing points depending on the durability, availability, I/O throughput (e.g., number of reads/writes per second), and latency. The most critical metric of offering cloud storage service is the data durability and availability. Due to the scale of operations, failures (both hardware and software) are not exceptions, but rather a normal operational characteristics seen on a daily basis in massive data centers. To cope with this challenge, data stored in the cloud is replicated in different availability zones. Considering a large fraction of data stored in the cloud are rarely accessed (e.g., once every six months), the added cost of any operational inefficiency is substantial. To reduce this cost burden, cloud operators such as AWS provide different storage options that provide reduced durability (e.g., using less replication) or availability (e.g., archiving large amounts of data by compressing and storing in tape storage). Since performance (e.g., high throughput, low latency, low jitter) is of paramount importance for some services, cloud operators also provide very fast storage mediums (e.g., database stored in memory with high network capacity) at premium pricing for storing and accessing data. Thus, users of cloud storage can segment their data based on the access patterns and optimize the average performance under their budget constraints by taking advantage of multiple tiers of cloud storage.

In this book, we try to abstract away various technology specific details and rather focus on few fundamental tradeoffs in cloud storage.

The first tradeoff that we will investigate exists between storage efficiency and network bandwidth required to maintain a target durability or availability in cloud storage. Existing results in the literature states that when we maximize the storage efficiency, the burden in terms of network bandwidth is also maximized. To alleviate the burden on network bandwidth (which is typically the most expensive resource), one must give up on storage efficiency. We will cover this tradeoff in detail first starting with the fundamental tradeoff from the point of information theory and then relaying results established with more practical considerations.

The second tradeoff that we will investigate exists between I/O latency vs. I/O throughput. An analogous tradeoff exists in classical communication theory between communication delay and communication capacity at a given loss rate. The main difference in the data center set up is that the network is considered reliable as applications typically run over reliable transport protocols. In cloud storage, the randomness of latency per I/O operation rather than packet losses is of primary concern. When data is replicated in more than one location, one can use more network bandwidth to retrieve the same data from multiple locations to reduce latency similar to the diversity techniques used in wireless communications. When the network bandwidth is finite, one has to tradeoff between I/O latency and I/O throughput, i.e., to achieve lower delay, one must serve less I/O requests. Since optimal operations require coded storage, new queue models that are not subject of classical textbooks on queuing theory emerge. We will cover these new types of queue models and provide various approximate as well as exact results established in the literature. Optimal solutions that can achieve the lowest latency at a targeted system throughput will also be presented. Since these solutions not only improve the average latency but also practically eliminate tail events, a much more predictable and reliable performance can be attained over storage clouds.

Next, we overview widely used techniques to address availability, throughput, and latency constraints.

1.1 Techniques to Improve the Performance of Distributed Storage

1.1.1 Striping and Parallelization

Striping refers to dividing the block or object to be stored in mutually exclusive smaller slices and distributing them to multiple storage nodes. When each storage node has its own local controller and can independently be accessed to retrieve its local slice, striping multiplexes the I/O throughput of multiple storage nodes by accessing all stripes of the block/object simultaneously in parallel. The stripe size must be picked carefully. If it is set to a small value, then I/O overheads such as actuating and rotating the magnetic discs, minimum round trip time delays over the network, etc. would be dominant terms reducing the throughput of individual nodes. If it is set to a large value, then the I/O capacity of fewer storage nodes are multiplexed. Since a successful I/O operation requires reading/writing all the stripes, the effective I/O speed is dominated by the slowest storage node. Although using small stripe size renders I/O speeds of individual stripes faster on average, it also increases the chances of hitting a slow storage node.

1.1.2 Replication

Replication refers to the process of creating multiple copies of the same block or object on distinct storage nodes. If striping is used, then each stripe is also effectively copied onto distinct storage nodes. Replication is relied upon heavily by well-known nosql database and file storage systems such as Hadoop File System (HDFS), Dynamo, Amazon S3, Memcached/Membase, Couchbase, Cassandra, etc. Many of these systems use default replication factor of 3, though many of these solutions allow replication factor to be set on a per file basis by their clients. Replication at the expense of using substantially more storage (e.g., $3\times$) leads to high data availability and durability. It also enables load balancing and load localization. One can define different availability zones, where an availability zone can be a physical server, rack of servers, different parts of a data center, different data centers, etc., to ensure that each copy/replica is placed in different availability zones.

1.1.3 Load Balancing

Load balancers try to distribute the load to individual servers (in our case storage nodes) to utilize all the available resources and keep average access latency at the lowest level possible.

Load balancing can be built into the system to ensure more even distribution of the incoming workload by randomization. A simple and effective way of such built in load distribution can be achieved through random hashing as follows. The first step is to map the object names (i.e., *keys*) that can be arbitrary strings into a fixed number of bits (e.g., 128-bit, 256-bit) using a well-known hash function. The hash value of the object represents the position of the object on a logical ring, where two hash values obtained by adding and subtracting 1 to and from this value correspond to the clockwise and counter clockwise neighboring points on the same logical ring. The next step is to partition the key space across the storage nodes. Each storage node i can be assigned t_i random hash values (called *tokens*) on the same logical ring. To identify the node that stores the object with a particular key is found by walking the logical ring starting from the target key in clockwise (or counter clockwise) direction until the first token is encountered. The node that owns this token stores the object. When number of tokens are sufficiently large and the value of t_i 's are set proportional to the resources of each node i , we can have even distribution of load without incurring large load deviations. When object j is replicated N_i times, instead of the first token encountered, first N_i tokens on the logical ring that owned by distinct storage nodes can be set as the replication locations. If it is desired to have multiple availability regions, each region can be organized as a separate logical ring. This type of logical ring organization goes back to early peer to peer (P2P) systems such as Chord [54] and is applied in modern systems such as Cassandra [34] and Dynamo [12].

Load balancers can also be built as programmable appliances, where candidate locations are assigned weights and a weighted round robin scheduling is performed by the load balancers. The weights are dynamically adjusted based on the load reports or latency measurements obtained from each node. As load balancers must monitor the per-

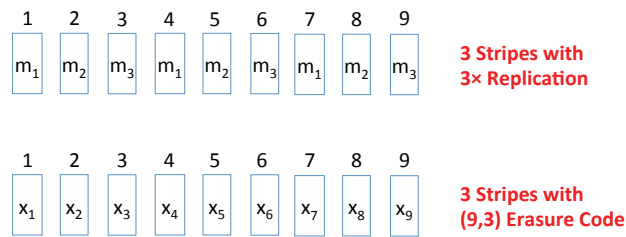


Figure 1.1: Data Replication vs. Coding.

formance of storage nodes and all requests are channeled through them, they have the potential to become bottlenecks. Therefore, as a lighter weight solution, load balancers often are designed for indirecting/routing requests to the relevant storage nodes, which in return directly communicates with the client application.

1.1.4 Coded Storage

Error correction codes have been used in individual memory hardware to detect and correct errors due to noise as well as the interference from the neighboring memory units for decades. In distributed storage systems, as described earlier, replication has been widely used to increase availability and data durability against losses of entire storage nodes. It is relatively recent that erasure coding is being utilized in large scale distributed storage systems (e.g., Microsoft Azure [23]). Coded storage is superior in terms of data reliability against simple replication when both techniques use the same amount of memory. In many situations it is also better in terms of I/O speeds as it allows maximum degree of freedom in accessing the data. Figure 1.1 illustrates this point in a toy example, where there are nine storage nodes storing a single file partitioned into three stripes m_1, m_2 , and m_3 . Via $3\times$ replication, each stripe is copied over three distinct nodes. In contrast, coded system generates nine encoded blocks x_1 through x_9 from original stripes m_1 through m_3 , then distribute them over the nine nodes. In the replication based system, storage client can use $3^3 = 27$ different subsets of nodes to retrieve the file, whereas in the coded system the number of choices

increases to $\binom{9}{3} = 84$ as any three nodes can be used to reconstruct the original file. For instance, a load balancer could pick the least loaded or closest three nodes to fetch the file without facing the constraints replication based system endures. The downside is that encoding/decoding overheads are introduced into the system as well as a more complicated repair procedure when failures occur almost inevitably. Coded storage is the main subject of our book and will be covered in details in the next two chapters.

1.1.5 Consistency Model

Distributed systems are designed with a consistency model in mind, which itself impacts the I/O performance. Data consistency becomes an issue when there are concurrent reads and writes for the same object. When there are N coded/uncoded copies of an object, a read or write request can be committed after R or W nodes ($1 \leq R, W \leq N$) that have a copy respond first, respectively. Picking parameters R, W such that $(R + W) > N$ ensures that the system is strongly consistent. In other words, any client is ensured that every read returns the latest version committed by the system. Under strong consistency constraint, setting R small increases W and vice versa. Thus, for read-heavy workloads it is desirable to set R to a low value, if possible to one. For write-heavy workloads, the reverse configuration becomes the optimal. For mixed workloads, one can optimize for the average or other moments of the overall delay distribution. Depending on how copies are located in terms of their topological or geographical distances, consistency model can even be used as a side information to perform better load balancing [32].

If the updates are strictly ordered among N nodes and clients attempt to read the copies following the same order, then again strong consistency can be ensured although $(R + W) \leq N$. The caveat is that we cannot balance the load across N copies and the system is designed mainly for data reliability.

Setting $(R + W) \leq N$ leads to eventual consistency when a client can be requesting the copy from any of the N nodes, i.e., the client will get the latest version of the object "eventually" for sure after all

the copies are updated. As a result, one can set $R = W = 1$ in this consistency model, increasing the I/O performance substantially for all workloads.

1.2 Distributed Storage Systems and Operational Characteristics

Distributed storage systems span a wide range of technologies and operational characteristics. From the bare-bone features such as how the storage nodes are topologically laid out and physically inter-connected (optical, coaxial, copper) to higher level features such as physical proximity of these nodes, their logical organization (e.g., centralized or decentralized), and the stack of communication protocols are being used (e.g., proprietary, Ethernet, TCP/IP, etc.) between the nodes can be quite different. The goal of this section is to highlight operational characteristics and techniques being employed to increase the performance, rather than presenting a detailed technology survey of past and existing distributed storage systems.

1.2.1 Disk Arrays

Magnetic disks have been the most cost effective way of storing large amounts of data. Rather than using one very expensive, high capacity and reliable disk, building disk arrays from much cheaper disks provides a solid alternative at a reduced cost with higher energy efficiency, throughput, and reliability when properly designed [43]. Redundant Arrays of Inexpensive/Independent Disks (RAID) systems become available in 1980s and by late 1990s they become the most common form of bulk storage in enterprise servers. Different RAID levels are defined as standard forms and they encompass various combinations of striping, replication, and coding. RAID 0 systems use striping for increased throughput, but does not provide protection against individual disk failures. RAID 1 uses replication (also called mirroring) without striping and coding. RAID 2, RAID 3, and RAID 4 provide bit, byte, and block level striping with a dedicated parity disk. RAID 5 and RAID 6 use single and double parity erasure codes without dedicated parity

disks, respectively. Nested RAID systems can also be built with different levels supporting different combinations of striping, replication, and coding.

Disk arrays are typically local (i.e., directly attached) to one physical server and their scale is relatively much smaller than the other distributed storage systems which can be collection of tens to millions of servers. The main bottleneck in magnetic discs is due to the involvement of mechanical components (e.g., to rotate the disk to access the right sector). They enjoy dedicated high bandwidth bus interfaces to directly connect to the host controller serving the host system. Coding is used mainly for higher degree of protection against failures rather than speeding up the I/O performance. On the contrary, when parity symbols must be written/read, the I/O throughput reduces.

1.2.2 Network Attached Storage (NAS) & Storage Area Network (SAN)

Both NAS and SAN systems provide access to a shared pool of disks that are not directly attached to the host computers, where applications are running. The main difference for the host servers is in the abstraction: NAS exposes itself as a file server hiding the details of the block storage underneath whereas SAN exposes itself as a block storage device that can be mounted as a disk volume and client must use volume management and formatting tools supported by its local operating system.

NAS systems can consist of one storage server with multiple disks on it or a cluster of such storage servers. In its cluster form, a distributed file system must be installed over the storage servers. Data can then be striped, replicated, and coded to be stored over multiple servers as in the RAID systems. The communication between the host computers and NAS servers occur over the local area network (LAN), which can be dedicated to the storage tier or shared with all the hosts in the network.

SAN systems typically use dedicated hardware and network for high performance. Fiber Channel, Ethernet, even IP networks can be used to connect the storage servers to the host computers. The interfaces to access block storage devices such as ATA and SCSI are communicated

over these networks or point-to-point channels with corresponding protocols such as FCP, AoE, FCoE, iFCP, iSCSI, etc.

Unlike directly attached disk arrays, NAS and SAN systems provide a shared and unified storage tier. The typical usage mode is limited to the enterprise boundaries and applications. Once deployed, these systems are relatively hard to expand by the services using them as new hardware must be installed requiring manual labor beyond simple configuration changes.

1.2.3 Peer to Peer (P2P) Storage

P2P systems have been quite popular since the late 1990s. P2P systems pool the computers of end users to build a service infrastructure by very small capital expenditure by the service providers. Peers are both the servers and consumers of the data stored in the P2P system. Such systems were experimentally as well as commercially built for non real-time file sharing services as well as real-time services such as voice over IP (VoIP), TV and video streaming. P2P systems pose the ultimate challenge for building a global-scale service infrastructure. As they are built over end user computers, which leave and join the network arbitrarily at will, they must be very resilient against frequent topology changes and node churn. Furthermore, end computers might have quite heterogeneous storage capacity and access bandwidth committed to the system. By their very virtue, P2P systems are built to operate over Internet. Therefore, the best effort traffic between peer nodes would see contention from the rest of the Internet.

P2P storage or file sharing systems have quite significant variations. In terms of content search, P2P systems typically utilized one of these approaches. (i) *Centralized server/tracker*: Peers publish their content and send their queries to this central server. (ii) *Super nodes*: Peers publish their content and send their queries to a super node. Queries are flooded among super nodes. A peer can be promoted to super node status depending on their lifetime in the system and available resources. (iii) *Query routing*: Peer queries are propagated via neighboring peers. In query routing, flooding, random walk, bloom filters, distributed hash tables (DHT) are among the techniques being heavily used.

Although search has been a very critical component, how to fetch the data at scale and reliably have been the main emphasis in practical systems. BitTorrent [44] is widely popular due to its success in delivering high performance in file download. It divides larger files into chunks of fixed size (e.g., 256KByte). A downloading peer requests missing chunks of a file from all the peers it is connected to (e.g., 20 peers) in the same torrent. Thus, different chunks are fetched from different peers in parallel. BitTorrent employs a rate control policy such that an uploader limits the number of its concurrent downloaders to a fixed number. Each uploader chokes its worst downloader (i.e., the one with the least upload rate) and serves (referred to as unchoking) another interested peer for the next period.

Erasur coding is also utilized in the context of P2P networks to prevent stalls due to rare pieces and facilitate reduced file download times. Particularly random network coding is proposed and evaluated in Avalanche system [19], where peers encode all incoming chunks/packets using linear codes before forwarding to the next peer. Even in topologies where network coding has no advantage, it simplifies the resource optimization problems to the extent where intractable problems with simple routing (e.g., NP-hard tree packing problems) can be transformed into tractable problems [9].

1.2.4 Cloud Storage

Cloud storage has many different flavors including distributed file systems supporting massive data storage and analytics (e.g., Hadoop Distributed File System (HDFS)), non-relational (i.e., NoSQL) databases (e.g., Amazon Dynamo DB, Microsoft Azure Redis Cache and DocumentDB, Google Cloud Datastore, Cassandra, MangoDB, Couchbase, etc.) supporting relatively small data, relational (i.e., SQL) databases, key-value stores for relatively large files (e.g., Amazon S3, Microsoft Azure Blob Storage, Google Cloud Storage, etc.), and block storage (e.g., Amazon EBS).

The main premise of the cloud storage is utility-based computing, storage, and networking. As demand increases or reduces on any resource tier, service provider can scale up or down that tier and pay

proportionately more or less. In the case of storage, pricing covers many things including actual storage space occupied (e.g., monthly average usage in the granularity of gigabytes), the number of reads and writes (e.g., every cent provides 10,000 reads or 1,000 writes, etc.), the amount of bytes transferred into and out of the cloud (e.g., free for transfer ins and per gigabyte charging for transfer outs), throughput guarantees (e.g., number of I/O operations per second), etc. Service level agreements can be different depending on the storage type. For database systems, delay and throughput guarantees are critical. Thus, throughput limits can be explicitly defined and charged for. For bulk storage, data persistency guarantees are more critical. Therefore, they can be explicitly defined and charged for. An example would be reduced data durability services in Amazon S3 and Azure Blob Storage that still provides significantly high durability guarantees (e.g., 10^{-9}) at a fraction of the cost of standard service (e.g., 10^{-11}). Next we provide few cloud services and their underlying storage paradigms.

Hadoop [53] and its various flavors have become one of the most important data analytics infrastructure that enable massive parallel computing. Hadoop is the open source version of Google's MapReduce [11] and it runs over Hadoop Distributed File System (HDFS). HDFS separates the cluster nodes into NameNodes and DataNodes. NameNodes store metadata of the files that includes information about blocks of the file and where each block is replicated. DataNodes are the servers that actually use their local disks to store file blocks. Files are divided into large blocks (e.g., of size 128 Mbyte) and each block is replicated on multiple DataNodes. Different blocks of the same files can be replicated to distinct servers. By default, each block is replicated to three DataNodes. When an HDFS client wants to write a new block of a file, it contacts its NameNode and asks for the DataNodes. Then, the client prepares a pipeline for replication and writes the block to the first DataNode in the pipeline. Each DataNode sends the replica block to the next DataNode on the pipeline. When an HDFS client wants to read a file, it contacts the NameNode to learn which blocks of the file is stored in which DataNodes. The client then directly contacts one of the NameNodes for each block (e.g., typically the closest NameNode

to the client). Hadoop distributes the MapReduce tasks to the machines closest to the data blocks they process. When blocks are placed on distinct servers, it enables parallel processing. Due to the replication, when many MapReduce jobs coexist, scheduler have more options to distribute the load evenly avoiding congestion on a DataNode that might have blocks that are concurrently processed by different tasks.

Dynamo DB is a public non-relational database offering by Amazon that can be used to store a structured table with items of small size (1 byte to 400 Kbyte). It combines the features from original Dynamo system [12] and SimpleDB offering again by Amazon. It stores the data in solid state disks (SSDs) and replicates each table onto three different availability zones that share no critical systems together (e.g., power supply, cooling, etc.). Within the same availability zone, one can read/write data items in the order of milliseconds. When clients use Simple Hash Key, a zero-hop distributed hash table (DHT) approach is used to partition the table items (see Section 1.1.5). When clients use Composite Hash Key with Range Key, composite hash attribute uniquely maps to the item location, while range attribute is used to retrieve all records within the range. In this case, all the items matching to the same composite hash key can be accessed from the same server.

Simple Storage Service (S3) is yet another Amazon service that can store data up to 5 Terabytes. It is an unstructured storage service, where unique keys (e.g., file names) are used to access each object. From S3 point of view, an object is simply a byte string of a predetermined length and it is the responsibility of the developer to organize/partition its data set to store on S3. S3 is quite flexible and one can design disk back up services, cloud based file services (e.g., DropBox), media delivery services (e.g., Netflix), or even overlay database services. S3 replicates the data to 3 different availability zones by default every time a new object is added. Since dense storage media (e.g., magnetic disks or even tapes for archival data) are utilized, the performance can be in the order of 10s of milliseconds even for small objects (e.g., 1Kbyte) within the same availability zone.

1.3 Book Organization

As visible from various different cloud offerings, the technologies and application domains have a lot of variety. In the following chapters, we will narrow our attention to the applications of erasure codes in the context of cloud based storage. Erasure coding is a critical piece of technology that plays a substantial role in the evolution of next generation cloud storage systems. Specifically, erasure codes have been proposed, prototyped and in some cases commercialized (e.g., HDFS-RAID originally developed by Facebook, Xorbas Hadoop [48], Amazon S3 [35]) as an enabling technology in cloud-based storage for increased efficiency or latency performance. Erasure codes on one hand can facilitate great reductions in storage costs and on the other hand can also bring about a more predictable, superior read/write (simply referred to as I/O) performance in the face of instantaneous, random performance degradation in parts of these massive systems.

We organized the remaining part of the book in two main chapters such that each chapter is self-contained and can be read independently covering key concepts in recent yet vast literature on how to design and use erasure codes in storage clouds. In Chapter 2, we cover erasure codes from the storage efficiency and network bandwidth points of view for a targeted protection against node failures. More specifically, we overview maximum distance separable codes (MDS), rateless codes, and repair/regenerative codes. The design of erasure codes with desired storage and bandwidth efficiency is the main topic of interest in this first main chapter. In Chapter 3, we turn our attention to the utilization of erasure codes as a tool to speed up the access latency for data stored in the cloud. Rather than focusing on the design of a code, we will assume a generic MDS code and look into the queueing dynamics, trade offs for accessing erasure coded data. Results derived both for purely mathematical models such as exponential service times and for more practical models derived from actual measurements on storage clouds are presented in this chapter.

2

Data Durability/Availability, Storage Efficiency, and Network Capacity

High data durability in storage systems is a measure of how likely data is lost once it is committed to the storage system. High availability in storage systems is a measure of the fraction of the time the storage system is operational and accessible by the clients. The targeted performance values in public clouds are quite high, e.g., an object loss rate of 10^{-11} per year and a downtime ratio of 10^{-4} corresponding to 52 minutes of total downtime annually. Although they measure different performance concerns, the main solution to provide both high durability and availability is the same: Prevent single point of failures and create sufficient degree of redundancy in the system.

Replicating a file on multiple physically distinct storage nodes in one or more datacenter locations is the norm for providing storage redundancy. Large files are divided into smaller chunks (k chunks) and are individually replicated on $R_{rep} = \frac{n}{k}$ distinct servers. Here, R_{rep} denotes the *replication factor* and n is the total number of stored chunks. When a particular chunk is requested, as long as any of the R_{rep} copies is accessible, then the system is available and can serve the request. Similarly, as long as any of the R_{rep} copies survives upon system failures, the original data is not lost. Providing redundancy in this form is

quite suboptimal to provide high availability and durability at a given utilization of storage space. Yet, it is simple to manage the system (e.g., simple copy operation to recover failed nodes is sufficient) and load balancing can be done easily without incurring extra overheads of data recovery improving the system latency.

In contrast, erasure coding provides a higher degree of redundancy by making replicas useful to recover more than one chunk of a given file. In its most storage efficient form, Maximum Distance Separable (MDS) codes ensure that accessing any k chunks out of n chunks would be sufficient to recover the original file. Thus, MDS codes survive arbitrary $(n - k)$ chunk failures as opposed to simple replication strategy that can sustain $(R_{rep} - 1) = (\frac{n}{k} - 1) = \frac{n-k}{k}$ failures. In other words, MDS codes can survive k -fold more chunk failures (equivalently, k -fold more server failures if each chunk is stored in a distinct server) than a brute-force replication strategy for the same storage cost.

Although MDS codes are superior in terms of requiring the least storage capacity for targeted availability and durability, they require a more complex maintenance: E.g., when a node fails, a simple copy operation is not sufficient and chunks from multiple nodes (in the worst case k nodes) must be fetched to repair the lost chunk. In addition, during the normal operations serving the client requests, the decoding delays must also be considered if non-systematic chunks are utilized.¹ Therefore, non-MDS constructions have also been considered in practice. In this chapter, we first start with linear MDS codes and present a linear algebraic view on its structure. Then, we overview an important class of non-MDS codes, i.e., *fountain/rateless codes*. Lastly, we switch to the main focus of the chapter, namely *repair codes* as these family of codes are critical to apply erasure codes to not only archival data, but also to data sets/files that are accessed more frequently.

¹Erasure codes can be systematic or non-systematic. In the systematic case, the first k symbol blocks/chunks generated by the erasure code are the same as the symbol blocks/chunks of the original file and referred to as *systematic* blocks/chunks. The remaining $(n - k)$ blocks/chunks are formed as (linear or non-linear) functions of these *systematic* blocks/chunks and referred to as *parity* blocks/chunks. In the case of non-systematic codes, some or none of the original blocks/chunks are the same as any of the encoded blocks generated by these erasure codes.

2.1 MDS Codes

This family of codes are optimal as any k blocks among n blocks would be sufficient for recovering the requested file which is of size k blocks. Suppose we operate on a finite field of size $q = 2^m$ and hence can be also represented by m bits. Finite fields are referred to as Galois fields named after their discoverer and we use the notation $GF(q)$ to denote a field of size q . Any (n, k) linear code can be put into a matrix product form as:

$$x = Gm, \quad (2.1)$$

where m is $k \times 1$ dimensional column vector whose elements are comprised of distinct symbols from the original file to be encoded, G is $n \times k$ dimensional generating matrix, and x is $n \times 1$ dimensional column vector corresponding to the encoding symbols. For a systematic code, the generating matrix takes the form:

$$G = \begin{bmatrix} I_{k \times k} \\ A \end{bmatrix},$$

where $I_{k \times k}$ is $k \times k$ dimensional identity matrix and A is $(n - k) \times k$ parity generating matrix. A nonsystematic generating matrix in the form $[B \ A]^t$ can be put into systematic form by post-multiplying it with the $k \times k$ matrix B^{-1} . The code properties are preserved as the whole system can be interpreted as passing a message vector $B^{-1}m$ through the non-systematic code and thus the codewords are the same. I.e.,

$$x = \left(\begin{bmatrix} B \\ A \end{bmatrix} B^{-1} \right) m = \begin{bmatrix} B \\ A \end{bmatrix} (B^{-1}m).$$

The files larger than k symbols can be sliced into mutually exclusive blocks of k symbols and encoded separately into n symbols. Figure 2.1 illustrates how such a slicing can be done. File is divided into α blocks, each comprised of k symbols. Each block (shown in the same column) of source symbols $x_{i,j}$, $j = 1, \dots, k$ are passed through the same ² systematic erasure code to generate parity symbols $x_{i,j}$, $j = (k+1), \dots, n$. Then, each row of α symbols are stored in distinct storage nodes.

²We can also use different erasure codes for each block

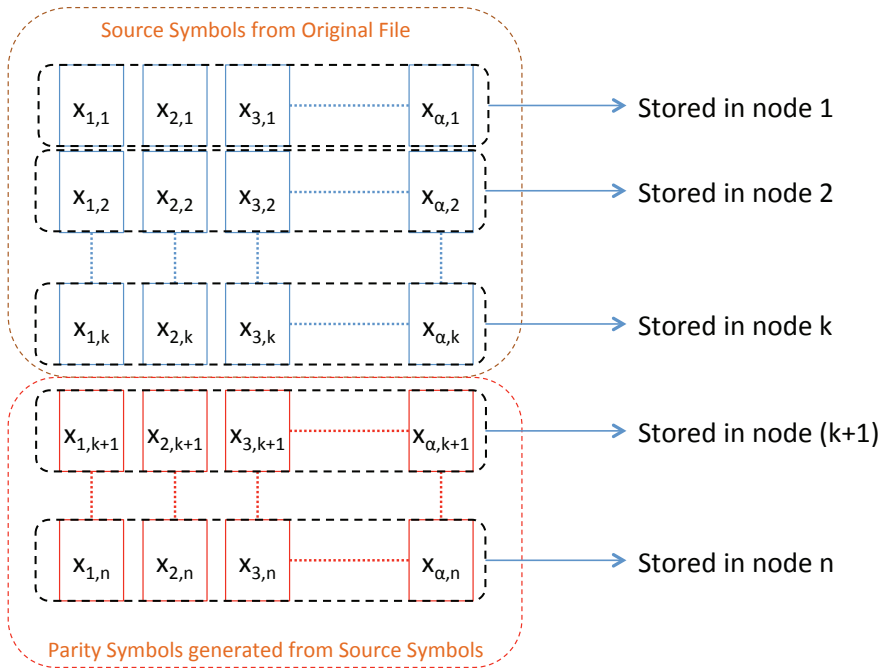


Figure 2.1: Illustration of how the original file can be divided into symbols, expanded using systematic erasure codes, and how encoded symbols are grouped together before transferred to the storage nodes.

To satisfy the MDS property, a necessary and sufficient condition is that any $k \times k$ submatrix of G should be a full rank matrix. One can randomly generate candidate matrices over a finite field (e.g., $GF(q)$) until this property is satisfied. Below, we provide few well-known deterministic constructions with the MDS property.

Parity codes are the most trivial MDS codes. An $(n, n - 1)$ parity code simply computes the bitwise XOR of source symbols to output a single parity symbol. Clearly, the original file can be recovered from any $k = (n - 1)$ nodes as at least $(k - 1)$ of them correspond to the original source symbol. In the case that we access the parity symbol as the k -th symbol, we can XOR all the received symbols to generate the missing source symbol.

Reed-Solomon (RS) codes are maybe the most famous family of non-binary channel codes. They are cyclic codes (i.e., every cyclic shift of a given codeword is also a codeword) and they satisfy the MDS property. Using symbols from $GF(q)$, an (n, k) RS code can be designed with $n = (q - 1)$. Characteristic polynomial $\mathbf{g}(X)$ of RS codes is given by

$$\begin{aligned}\mathbf{g}(X) &= (X - s)(X - s^2) \cdots (X - s^{n-k}) \\ &= g_0 + g_1X + g_2X^2 + \cdots + g_{n-k}X^{n-k}\end{aligned}$$

where $g_i \in GF(q)$, $i = 1, \dots, (n-k)$ and s^i 's are roots of the polynomial $X^{q-1} - 1$. The generating matrix in (2.1) can be written in transpose form as follows:

$$G^t = \begin{bmatrix} g_0 & g_1 & \cdot & \cdot & \cdot & \cdot & \cdot & g_{n-k} & 0 & 0 & 0 & \cdot & 0 \\ 0 & g_0 & g_1 & \cdot & \cdot & \cdot & \cdot & \cdot & g_{n-k} & 0 & 0 & \cdot & 0 \\ 0 & 0 & g_0 & g_1 & \cdot & \cdot & \cdot & \cdot & \cdot & g_{n-k} & 0 & \cdot & 0 \\ \vdots & & & & & & & & & & & \vdots & \\ 0 & 0 & 0 & \cdot & \cdot & 0 & g_0 & g_1 & \cdot & \cdot & \cdot & \cdot & g_{n-k} \end{bmatrix}$$

Since any MDS code can be truncated to $k \leq n' < n$ from an existing (n, k) MDS code, we can first construct an RS code and then obtain a truncated MDS code. The resulting MDS code however no longer preserves the cyclic property and is not an RS code.

Cauchy matrices constitute a rich family of matrices that satisfy the MDS property. For a generating matrix G to be a Cauchy matrix, its entries g_{ij} (i.e., i -th row and j -th column element) satisfy the following relation:

$$g_{ij} = \frac{1}{a_i - b_j} \quad \text{s.t. } (a_i - b_j) \neq 0, \quad 1 \leq i \leq n, \quad 1 \leq j \leq k,$$

where a_i and b_i are elements of a finite field of size at least n and $a_i \neq a_j$, $b_i \neq b_j$ for $i \neq j$. Every square Cauchy matrix has full rank and any submatrix of a Cauchy matrix is also a Cauchy matrix. Thus, any k rows of G would form a $k \times k$ Cauchy matrix that is full rank.

Vandermonde matrices form another family of matrices that can be used to construct generating matrices with the MDS property. An

$m \times l$ Vandermonde matrix V is defined as follows:

$$V = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ a_1 & a_2 & a_3 & \cdots & a_l \\ a_1^2 & a_2^2 & a_3^2 & \cdots & a_l^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_1^{m-1} & a_2^{m-1} & a_3^{m-1} & \cdots & a_l^{m-1} \end{bmatrix}.$$

Given all its elements a_i are distinct, Vandermonde matrix is of full rank. However, submatrices of a Vandermonde matrix is not another Vandermonde matrix in contrast to what we have in the case of Cauchy matrices. Nonetheless, we can use two Vandermonde matrices to construct a systematic MDS code according to the results by Lacan & Fimes [33]. Let V_1^t and V_2^t are two $k \times k$ and $k \times (n - k)$ Vandermonde matrices with mutually distinct elements. Then, the following generating matrix in systematic form satisfies the MDS property:

$$G = \begin{bmatrix} I_{k \times k} \\ V_2(V_1)^{-1} \end{bmatrix}.$$

MDS codes by their very constraint cannot be too sparse. In fact, the parity symbols must bear information about every source symbol as otherwise accessing this single parity symbol along with $k - 1$ source symbols would not be able to recover the missing source symbol. On the other hand, sparser generating matrices would allow more efficient encoding/decoding as well as enable more localized repairs. Next, we look at an important class of such codes, namely Fountain/Rateless Codes.

2.2 Fountain/Rateless Codes

Rateless codes provide an infinite stream of coded symbols and they can recover the original symbols after receiving $(1 + \epsilon)k$ symbols, where $\epsilon > 0$. In this section, we provide an overview of well known linear constructions, namely LT Codes and Raptor Codes [3, 52]. There are also more recent constructions such as Strider and Spinal Codes that are mainly proposed for the error-prone and unpredictable wireless channels.

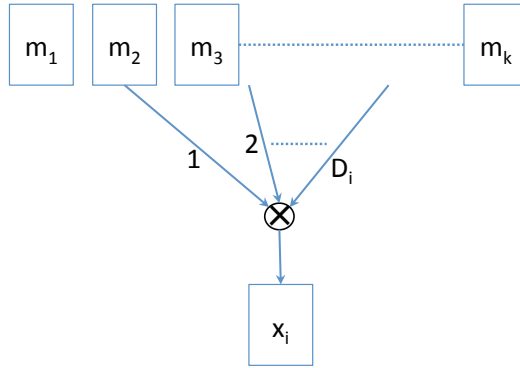


Figure 2.2: Coding Structure of LT Codes.

LT codes are the first rateless erasure codes [40]. They are linear codes that can be interpreted as a code that has a pseudo-random generating matrix with a low average density, an unbounded number of rows, and entries from $GF(2)$. Unlike deterministic constructions, recovering the original source symbols after receiving a minimum number of coded symbols becomes probabilistic. The coding structure of LT codes is shown in Figure 2.2. To generate i -th encoded symbol x_i , first a random number D_i is drawn from a well-known probability distribution e.g., robust soliton distribution (see the definition in [40]). Then, D_i source symbols are picked uniformly at random among k source symbols and XORED together to generate x_i . LT codes become more efficient (i.e., get closer to MDS codes) as k goes to infinity. Nonetheless, its performance suffers substantially even when k is in the order of one thousand. In LT codes, the only optimization one can perform is to design or search for a well-performing degree distribution. The degree distribution is biased towards lower degrees leading to much lower encoding/decoding complexity than MDS codes.

To boost up its performance, raptor codes are developed [52]. The source symbols are first encoded into intermediate symbols using a well designed deterministic precoder G_{pre} and then these intermediate symbols are passed through the LT Coder to generate as many encoded symbols as needed. The degree distribution in the LT Coder block is



Figure 2.3: Block Diagram of Raptor Codes.

carefully designed/searched and one such degree distribution is given by the polynomial:

$$\begin{aligned} \Omega(D) = & 0.007969D + 0.493570D^2 + 0.166220D^3 + 0.072646D^4 + \\ & 0.082558D^5 + 0.056058D^8 + 0.037229D^9 + \\ & 0.055590D^{19} + 0.025023D^{64} + 0.003135D^{66} \end{aligned}$$

In this polynomial, the coefficients are the probabilities and the exponents are the degree values, e.g., $0.493570D^2$ means the probability of $D = 2$ is 0.493570. G_{pre} can be optimized given the performance of the LT block for the given degree distribution. The overall block diagram is showing in Figure 2.3. Luby et al. [41] formulates the performance of raptor codes concisely as:

$$P_f(l, k) = \begin{cases} 1, & l < k \\ 0.85 \times 0.567^{l-k}, & l \geq k \end{cases},$$

where $k > 200$, $(l - k)$ is the measure of additional overhead with respect to an MDS code, and P_f is the probability of unsuccessful recovery of the original k symbols. Note that P_f is not a measure of partial recovery, even when only 1 symbol remains missing, the recovery is unsuccessful. From the formula, it can be directly seen that when k encoded symbols are received, the probability of success is merely 15%. With $k + 50$ encoded symbols, one can achieve 12 nines of reliability (or $P_f \leq 10^{-12}$). This is a higher durability guarantee than provided by any public storage services such as Amazon S3. For large number of source symbols, 50 symbol overhead becomes marginal and for all practical purposes raptor code performs almost like an MDS codes in erasure channels.

Both LT and Raptor codes are non-systematic codes due to the pseudo-random generating matrix. However, if pseudo-random matrix

is known a priori, then a systematic code can be constructed by taking any full-rank $k \times k$ sub-matrix of the pseudo-random generating matrix and pre-multiplying the source symbols with its inverse matrix similar to the case of MDS codes. The row indices of the full-rank sub-matrix determines the systematic positions in the encoded symbols. Since, (unlike MDS codes) not all $k \times k$ sub-matrices are full rank, we cannot arbitrarily pick any sub-matrix.

LT and Raptor codes are amenable to fast encoding/decoding implementation due to their low average degrees and minimum field size. Plus, one can potentially use all the storage nodes and later can progressively release memory to other files without hampering the minimum durability and availability guarantees. However, whether such constructions are useful or not when network bandwidth is the major bottleneck is not clear in the context of making local repairs to replace the lost storage nodes. We dedicate the rest of this chapter to the repair problem for bandwidth limited environments.

2.3 Regenerating/Repair Codes

MDS codes provide the lowest storage cost at a given reliability target. However, to maintain the same degree of reliability after one or more storage nodes fail, they need to be replaced by other storage nodes. The repair should be such that storage client (referred to as *data collector*) should still be able to contact any k nodes out of n storage nodes to recover the file it demands. As a brute-force method of repair, replacement nodes can simply act as a collector node: they contact any k nodes to reconstruct the original file and then regenerate the blocks of the failed nodes to store locally. Note that if file size is B symbols, this procedure transfers $k \times B/k$ symbols to replace the B/k symbols stored at each failed node. Thus, the repair bandwidth is k times more than the amount of information to be replaced. This excessive bandwidth used for repairs can be overwhelming for storage cloud providers as network bandwidth is the most limited resource. Note that the brute-force approach puts a very tight constraint on the repair problem, e.g., requiring any k nodes to be sufficient for the repair. It turns out that

the repair bandwidth can be significantly reduced by relaxing this constraint to "contact any $d \geq k$ surviving nodes to replace the failed node" [13]. We will refer to d as the repair degree.

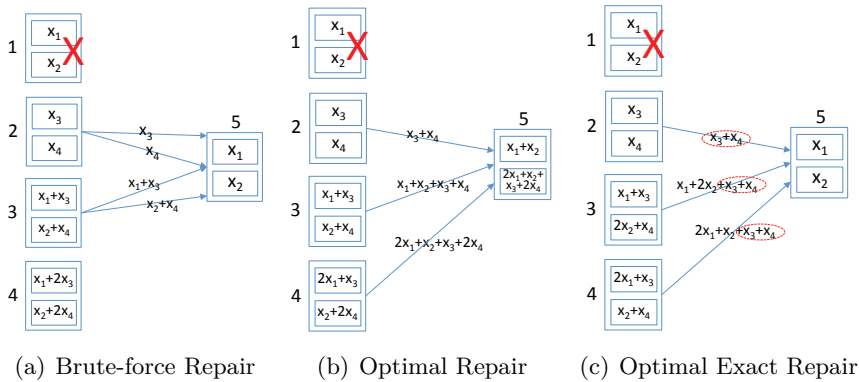


Figure 2.4: Toy examples to illustrate the differences in different repair codes.

Figure 2.4 provides several toy examples to help better understand the formal model introduced in the following section as well as several design choices. In this example, we have 4 storage nodes ($n = 4$), each node can store only 2 symbols and the file is comprised of 4 symbols (labeled as x_1 through x_4). Figure 2.4(a) provides how any MDS code can be used to repair the data on a failed node. The file is sliced into two stripes $\{x_1, x_3\}$ and $\{x_2, x_4\}$. Each stripe is then passed through a $(4, 2)$ -MDS code to generate two parity blocks to expand each stripe into 4 symbols. Each node stores one distinct symbol per stripe. When any node is lost, data can be exactly repaired by contacting any 2 nodes. In the figure, the example shows when node 1 fails and nodes 2 & 3 are used for the repair. Overall, 4 symbols are transferred for repairing x_1 and x_2 . Figure 2.4(b) shows a coding strategy that achieves the best possible repair bandwidth requiring 3 symbols to be transferred from surviving nodes to the replacement node. Note that each surviving node sends one symbol formed as a linear combination of the locally stored symbols. The replacement node (i.e., node 5) takes another linear combination of these 3 received symbols to compute the 2 symbols to