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

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 LaTeX using a class file designed by Neal Parikh. Printed on acid-free paper.

ISBN: 978-1-68083-065-1

Foundations and Trends® in Networking
Volume 9, Issue 4, 2014

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: 10.1561/1300000051

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

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: 10.1561/1300000051.

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 analagous 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

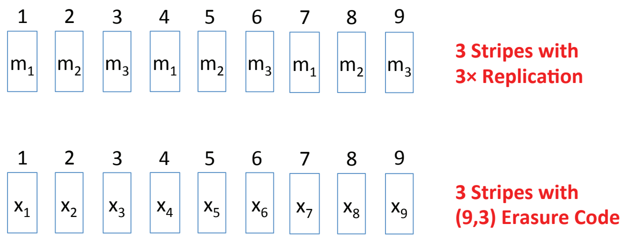
Replication refers to the process of creating multiple copies of the same block or object on distinct storage nodes. If striping is used, then each stripe is also effectively copied onto distinct storage nodes. Replication is relied upon heavily by well-known nosql database and file storage systems such 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-



1 2 3 4 5 6 7 8 9
m1 m2 m3 m1 m2 m3 m1 m2 m3
3 Stripes with 3× Replication
1 2 3 4 5 6 7 8 9
x1 x2 x3 x4 x5 x6 x7 x8 x9
3 Stripes with (9,3) Erasure Code

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× 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 numbed. 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.

Erasure 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, Couchbased, 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 hast 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.

References

[1] Rudolf Ahlswede, Ning Cai, Shuo-Yen R. Li, and Raymond W. Yeung Network information flow. *IEEE Transactions on Information Theory*, 46(4):1204–1216, Jul 2000.

[2] Paul J. Burke. The output of a queuing system. *Operations Research*, 4(6):699–704, 1956.

[3] John W. Byers, Gu-In Kwon, Michael Luby, Michael Mitzenmacher A digital fountain approach to asynchronous reliable multicast. *IEEE Journal on Selected Areas in Communications*, 20(8):1528–1540, Oct 2002.

[4] Viveck R Cadambe, Syed Ali Jafar, Hamed Maleki, Kannan Ramchandran, and Changho Suh. Asymptotic interference alignment for optimal repair of mds codes in distributed storage. *IEEE Transactions on Information Theory*, 59(5):2974–2987, 2013.

[5] Chris X. Cai, Guanfeng Liang, and Ulas C Kozat. Load balancing and dynamic scaling of cache storage against zipfian workloads. In *IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, 2014.

[6] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):4, 2008.

[7] Peter M Chen, Edward K Lee, Garth A Gibson, Randy H Katz, and David A Patterson. Raid: High-performance, reliable secondary storage. *ACM Computing Surveys (CSUR)*, 26(2):145–185, 1994.

[8] Shengbo Chen, Yin Sun, Ulas C. Kozat, Longbo Huang, P. Sinha, Guanfeng Liang, Xin Liu, and Ness B. Shroff. When queueing meets coding: Optimal-latency data retrieving scheme in storage clouds. In *Proceedings of INFOCOM 2014*, pages 1042–1050, Apr 2014.

[9] Dah Ming Chiu, Raymond W. Yeung, Jiaqing Huang, and Bin Fan. Can network coding help in p2p networks? In *4th International Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks*, pages 1–5, Apr 2006.

[10] Brian F Cooper, Raghu Ramakrishnan, Utkarsh Srivastava, Adam Silberstein, Philip Bohannon, Hans-Arno Jacobsen, Nick Puz, Daniel Weaver, and Ramana Yerneni. Pnuts: Yahoo!'s hosted data serving platform. *Proceedings of the VLDB Endowment*, 1(2):1277–1288, 2008.

[11] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, Jan 2008.

[12] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's highly available key-value store. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 205–220. ACM, 2007.

[13] Alexandros G. Dimakis, P. Brighten Godfrey, Yunnan Wu, Martin J. Wainwright, and Kannan Ramchandran. Network coding for distributed storage systems. *IEEE Transactions on Information Theory*, 56(9):4539–4551, Sept 2010.

[14] Alexandros G. Dimakis, Kannan Ramchandran, Yunnan Wu, and Changho Suh. A survey on network codes for distributed storage. *Proceedings of the IEEE*, 99(3):476–489, 2011.

[15] Alessandro Duminuco and Ernst W Biersack. Hierarchical codes: A flexible trade-off for erasure codes in peer-to-peer storage systems. *Peer-to-peer networking and applications*, 3(1):52–66, 2010.

[16] Toni Ernvall. Exact-regenerating codes between MBR and MSR points. *CoRR*, abs/1304.5357, 2013.

[17] Ulric J. Ferner, Tong Wang, and Muriel Medard. Network coded storage with multi-resolution codes. In *Signals, Systems and Computers, 2013 Asilomar Conference on*, pages 652–656, Nov 2013.

[18] Simson L. Garfinkel. An Evaluation of Amazon's Grid Computing Services: EC2, S3 and SQS. Technical report, Harvard University, 2007.

[19] Christos Gkantsidis and Pablo Rodriguez Rodriguez. Network coding for large scale content distribution. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 4, pages 2235–2245. IEEE, 2005.

[20] Sreechakra Goparaju, Salim El Rouayheb, and A. Robert Calderbank. New codes and inner bounds for exact repair in distributed storage systems. *CoRR*, abs/1402.2343, 2014.

[21] Kevin M. Greenan , Xiaozhou Li , and Jay J. Wylie. Flat xor-based erasure codes in storage systems: Constructions, efficient recovery, and tradeoffs. In *IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–14, May 2010.

[22] Tracey Ho, Muriel Médard, Ralf Koetter, David R Karger, Michelle Effros, Jun Shi, and Ben Leong. A random linear network coding approach to multicast. *IEEE Transactions on Information Theory*, 52(10):4413–4430, 2006.

[23] Cheng Huang, Huseyin Simitci, Yikang Xu, Aaron Ogus, Brad Calder, Parikshit Gopalan, Jin Li, Sergey Yekhanin, et al. Erasure coding in windows azure storage. In *USENIX ATC*, volume 12, 2012.

[24] Longbo Huang, Sameer Pawar, Hao Zhang, and Kannan Ramchandran. Codes can reduce queueing delay in data centers. In *IEEE International Symposium on Information Theory Proceedings (ISIT)*, pages 2766–2770. IEEE, 2012.

[25] Sidharth Jaggi, Peter Sanders, Philip A. Chou, Michelle Effros, Sebastian Egner, Kamal Jain, and Ludo M. G. M. Tolhuizen. Polynomial time algorithms for multicast network code construction. *IEEE Transactions on Information Theory*, 51(6):1973–1982, June 2005.

[26] Gauri Joshi, Yanpei Liu, and Emina Soljanin. Coding for Fast Content Download. In *50th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 326–333. IEEE, 2012.

[27] Gauri Joshi, Yanpei Liu, and Emina Soljanin. On the delay-storage trade-off in content download from coded distributed storage systems. *arXiv preprint arXiv:1305.3945*, 2013.

[28] Gauri Joshi, Yanpei Liu, and Emina Soljanin. On the Delay-Storage Trade-Off in Content Download from Coded Distributed Storage Systems. *IEEE Journal on Selected Areas in Communications*, 32(5):989–997, 2014.

[29] Osama Khan, Randal Burns, James Plank, William Pierce, and Cheng Huang. Rethinking erasure codes for cloud file systems: Minimizing i/o for recovery and degraded reads. In *USENIX FAST*, 2012.

[30] Cheeha Kim and Ashok K. Agrawala. Analysis of the fork-join queue. *IEEE Transactions on Computers*, 38(2):250–255, Feb 1989.

[31] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. Zyzzyva: Speculative Byzantine fault tolerance. *ACM Transactions on Computer Systems*, 27:7:1–7:39, 2010.

[32] Hiroyuki Kubo and Ulas C Kozat. On improving latency of geographically distributed key-value stores via load balancing with side information. In *IEEE International Conference on Communications (ICC)*, pages 3710–3715. IEEE, 2013.

[33] Jerome Lacan and Jerome Fimes. Systematic mds erasure codes based on vandermonde matrices. *IEEE Communications Letters*, 8(9):570–572, Sept 2004.

[34] Avinash Lakshman and Prashant Malik. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2):35–40, 2010.

[35] Guanfeng Liang and U.C. Kozat. TOFEC: Achieving optimal throughput-delay trade-off of cloud storage using erasure codes. In *Proceedings IEEE INFOCOM 2014*, pages 826–834, Apr 2014.

[36] Guanfeng Liang and Ulas C Kozat. Use of erasure code for low latency cloud storage. In *52nd Annual Allerton Conference on Communication, Control, and Computing, 2014*, Oct 2014.

[37] Guanfeng Liang and Ulas C Kozat. FAST CLOUD: Pushing the Envelope on Delay Performance of Cloud Storage with Coding. *IEEE/ACM Transactions on Networking*, preprint, 13 Nov. 2013. doi: 10.1109/TNET.2013.2289382.

[38] Guanfeng Liang and Ulas C. Kozat. On throughput-delay optimal access to storage clouds via load adaptive coding and chunking. *arXiv preprint arXiv:1403.5007*, 2014.

[39] Zimu Liu, Chuan Wu, Baochun Li, and Shuqiao Zhao. Uusee: large-scale operational on-demand streaming with random network coding. In *Proceedings IEEE INFOCOM 2010*, pages 1–9. IEEE, 2010.

[40] Michael Luby. LT codes. In *43rd Annual IEEE Symposium on Foundations of Computer Science*, pages 271–280, 2002.

[41] Michael Luby, Tiago Gasiba, Thomas Stockhammer, and Mark Watson. Reliable multimedia download delivery in cellular broadcast networks. *IEEE Transactions on Broadcasting*, 53(1):235–246, Mar 2007.

[42] Dimitris S Papailiopoulos, Jianqiang Luo, Alexandros G Dimakis, Cheng Huang, and Jin Li. Simple regenerating codes: Network coding for cloud storage. In *Proceedings IEEE INFOCOM 2012*, pages 2801–2805. IEEE, 2012.

[43] David A. Patterson, Garth Gibson, and Randy H. Katz. A case for redundant arrays of inexpensive disks (raid). *SIGMOD Record*, 17(3):109–116, Jun 1988.

[44] Dongyu Qiu and R. Srikant. Modeling and performance analysis of bittorrent-like peer-to-peer networks. In *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM 2004, pages 367–378, New York, NY, USA, ACM, 2004.

[45] K. Vinayak Rashmi, Nihar B. Shah, and P. Vijay Kumar. Optimal exact-regenerating codes for distributed storage at the MSR and MBR points via a product-matrix construction. *CoRR*, abs/1005.4178, 2010.

[46] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware 2001*, pages 329–350. Springer, 2001.

[47] Maheswaran Sathiamoorthy, Megasthenis Asteris, Dimitris Papailiopoulos, Alexandros G. Dimakis, Ramkumar Vadali, Scott Chen, and Dhruba Borthakur. Xoring elephants: novel erasure codes for big data. In *Proceedings of the 39th international conference on Very Large Data Bases*, PVLDB 2013, pages 325–336. VLDB Endowment, 2013.

[48] Maheswaran Sathiamoorthy, Megasthenis Asteris, Dimitris S. Papailiopoulos, Alexandros G. Dimakis, Ramkumar Vadali, Scott Chen, and Dhruba Borthakur. Xoring elephants: Novel erasure codes for big data. *CoRR*, abs/1301.3791, 2013.

[49] Nihar B. Shah, K. Vinayak Rashmi, P. Vijay Kumar and Kannan Ramchandran. Explicit codes minimizing repair bandwidth for distributed storage. In *IEEE Information Theory Workshop (ITW)*, pages 1–5, Jan 2010.

[50] Nihar B. Shah, K. Vinayak Rashmi, P. Vijay Kumar and Kannan Ramchandran. Distributed storage codes with repair-by-transfer and nonachievability of interior points on the storage-bandwidth tradeoff. *IEEE Transactions on Information Theory*, 58(3):1837–1852, Mar 2012.

[51] Nihar B. Shah, Kangwook Lee, and Kannan Ramchandran. The MDS queue: Analysing the latency performance of erasure codes. *arXiv preprint arXiv:1211.5405*, 2013.

[52] Amin Shokrollahi. Raptor codes. *IEEE Transactions on Information Theory*, 52(6):2551–2567, 2006.

[53] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–10. IEEE, 2010.

[54] Ion Stoica, Robert Morris, David Liben-Nowell, David R Karger, M Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking*, 11(1):17–32, 2003.

[55] Changho Suh and Kannan Ramchandran. Exact regeneration codes for distributed storage repair using interference alignment. *CoRR*, abs/1001.0107, 2010.

[56] Chao Tian, Birenjith Sasidharan, Vaneet Aggarwal, Vinay A. Vaishampayan, and P. Vijay Kumar. Layered, exact-repair regenerating codes via embedded error correction and block designs. *CoRR*, abs/1408.0377, 2014.

[57] Yunnan Wu, Alexandros Ros Dimakis, and Kannan Ramchandran. Deterministic regenerating codes for distributed storage. In *Allerton Conference on Communication, Control, and Computing*, 2007.

[58] Yu Xiang, Tian Lan, Vaneet Aggarwal, and Yih-Farn Robin Chen. Joint latency and cost optimization for erasure-coded data center storage. *arXiv preprint arXiv:1404.4975*, 2014.

[59] Zhe Zhang, Amey Deshpande, Xiaosong Ma, Eno Thereska, and Dushyanth Narayanan. Does erasure coding have a role to play in my data center. *Microsoft Research MSR-TR-2010*, 52, 2010.

[60] Ben Y. Zhao, John D. Kubiatowicz, and Anthony D. Joseph. Tapestry: A fault-tolerant wide-area application infrastructure. *SIGCOMM Computer Communations Review*, 32(1):81–81, Jan 2002.