Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2013, Article ID 756185, 11 pages
http://dx.doi.org/10.1155/2013/756185

Research Article

BMCloud: Minimizing Repair Bandwidth and Maintenance Cost in Cloud Storage

Chao Yin,[1] Changsheng Xie,[1,2] Jiguang Wan,[1,2] Chih-Cheng Hung,[3,4] Jinjiang Liu,[5] and Yihua Lan[5]

[1] *School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China*
[2] *Wuhan National Laboratory for Optoelectronics, Wuhan 430074, China*
[3] *School of Computer and Information Engineering, Anyang Normal University, Anyang 455000, China*
[4] *Center for Biometrics Research, Southern Polytechnic State University, Atlanta, GA, USA*
[5] *School of Computer and Information Technology, Nanyang Normal University, Nanyang 473061, China*

Correspondence should be addressed to Changsheng Xie; cs_xie@mail.hust.edu.cn

Received 11 July 2013; Accepted 8 September 2013

Academic Editor: William Guo

To protect data in cloud storage, fault tolerance and efficient recovery become very important. Recent studies have developed numerous solutions based on erasure code techniques to solve this problem using functional repairs. However, there are two limitations to address. The first one is consistency since the Encoding Matrix (EM) is different among clouds. The other one is repairing bandwidth, which is a concern for most of us. We addressed these two problems from both theoretical and practical perspectives. We developed BMCloud, a new low repair bandwidth, low maintenance cost cloud storage system, which aims to reduce repair bandwidth and maintenance cost. The system employs both functional repair and exact repair while it inherits advantages from the both. We propose the JUDGE_STYLE algorithm, which can judge whether the system should adopt exact repair or functional repair. We implemented a networked storage system prototype and demonstrated our findings. Compared with existing solutions, BMCloud can be used in engineering to save repair bandwidth and degrade maintenance significantly.

1. Introduction

With the rapid growth of data production in companies, the requirement of storage space grows very largely as well. This growth leads to the emergence of cloud storage. Cloud storage is a concept which is an extension and development from cloud computing. This system collects application software in order to work together and provide systems of data storage and business access features through grids or distributed file systems [1]. Cloud storage is produced by distributed storage technology and virtualization technology, and it is the latest development of distributed storage technology. Cloud storage provides effective solutions for network mass data storage. Also this system provides on-demand pay services, which reduces not only the threshold for the user but also the payment. Some companies have been involved in cloud storage systems, such as Amazon, Sun, Google, and Yahoo.

According to forecasts from the International Data Corporation (IDC), the size of the global cloud computing and cloud storage market has increased from \$16 billion in 2008 to \$42 billion in 2012 [2], which is the proportion of the total IT investment from 4.2% to 5%. In addition, the IDC predicts that cloud computing and cloud storage investments will be 25% of the annual IT investment in 2012, and that will rise to 30% or more in 2013. The advisory body Merrill (MerriliLynch) foresees that this will be \$160 billion market by 2011.

In order to improve the reliability of cloud storage, some companies have developed their own solutions, such as HDFS [3] and GFS [4]. The two cloud storage systems use duplicate technology to have reliability guaranteed. Of course, this method can solve the problem of data loss and data error. But there is a lot of storage space wasted by using this method and this will bring consistency issues. For these reasons, the erasure code has become one of the most popular choices for storage to improve its utilization and avoid the problem of consistency. A good erasure code technology can not only improve the availability and reliability of the system but can also improve the efficiency of data access.

We hereby focus on the recovery problem for a family of network coding. In the current situation, cloud coding is a popular direction to prevent large-scale cloud node failure. While bandwidth consumption is an important performance signal in cloud storage system, we always want to repair data using the minimum bandwidth and the fastest repair speed. The model for a cloud file system using erasure codes is inspired by NCCloud [5], which is the implementable design for the functional minimum-storage regenerating code [6]. Other cloud file systems such as Khan et al. [7] are also function repair systems. In functional repair, the system needs to keep the value of the repair matrix to ensure consistency. Following each functional repair, that matrix needs to be synchronized to each cloud in order to ensure the consistency across the system. The number of recovery matrices increases with the number of recoveries, which will affect the overall system reliability and consistency. Therefore we should establish a system with an exact repair in case the system experiences data loss or data error.

In this paper, we have proposed Bandwidth and Maintenance minimum Cloud system (BMCloud) which has low bandwidth consumption and low overhead in terms of maintenance. The system has both functional repair and exact repair when data loss or data error are experienced. When a part of data in cloud is lost, the system can recover with exact repair to degrade the overhead of maintenance for the future. More importantly, it has the ability to consume less bandwidth when recovering. While almost all the data in the cloud breaks, the system can recover it with a functional repair. We have developed a JUDGE_STYLE rule to judge whether the system should use exact repair to recover or not.

The system includes a proxy which can calculate and transfer data through the clouds and can maintain the system consistency. In the exact repair function, the proxy itself does not require arithmetic processing of the data or the data cache, and it does not need to provide calculation and storage capabilities. Since most of the calculation work is loaded on the cloud nodes in this function, which will be described in Section 4 with more details, the proxy would not be a bottleneck in terms of calculation and storage.

The contributions of this paper are described as follows.

(i) We have developed E Code algorithm to recover data in exact repair. It can improve recovery bandwidth performance and ensure data integrity.

(ii) We have developed the JUDGE_STYLE algorithm, which can judge whether the system should use exact repair or functional repair.

(iii) We have implemented the system BMCloud. When the number of the fail strips in the cloud is less than or equal to 4, or between 4 and 10 but we can recover data in exact repair, BMCloud could significantly improve repair bandwidth by 53.9% and 41.8% compared to current solutions (RDP, NCCloud, etc.). In addition, it can degrade maintenance cost.

The rest of this paper is organized as follows. Section 2 describes related work in erasure code. The architecture and the design of BMCloud are introduced in Section 3. Section 4 is the prototype and algorithms. Section 5 is the experimental result and evaluation. Section 6 is the conclusions.

2. Related Works and Motivation

2.1. RAID Code. Since Patterson et al. [8] brought the concept of RAID into storage systems, erasure coding has been one of the most popular choices to supply high reliability and high performance storage services with acceptable spatial and monetary cost. Examples such as LAN, RAID0, RAID1, and RAID5 are very small and their integration versions are widely used. These can be used to recover systems when a disk error occurs. With the reliability requirements improved, single-fault-tolerant could not meet the needs of the consumers, so two-fault-tolerant comes into play. There are a lot of two-fault-tolerant codes storage systems, such as EVENODD [9], RDP [10], and X-code [11]. A typical RAID-6 storage system is composed of $k + 2$ disk drives. The first k disk drives are used to store original data, and the last two are used as parity disk drives. All the disk drives are in two check chains. When one or two disk drives lose data, we can recover data from the two check chain. Wan et al. [12] have used the raid code to improve the system performance and save the energy.

Maximum Distance Separable (MDS) can tolerate maximum failures with a given amount of redundancy. It is widely used in RAID-6 code, so that most of the RAID-6 codes are MDS codes. The most important feature of MDS is the ability to verify that the length of the chain is equal to the length of all the disks, which means that all disks are involved in checking, so you can reach the maximum disk utilization. But there is another coding which is different to MDS. In these cases, the length of the chain is less than the length of all the disks, such as in M-code [13]. Therefore due to careful design of the parity chain arrangement, the length of its check chain is shorter than that of the conventional RAID-6 code, which makes its I/O consumption smaller.

2.2. Network Coding. With the development of the network, especially the development of the cloud in recent years, erasure code is also transferred from the disk array level to the cloud level. In network coding, there are three versions of repair which include exact repair, functional repair, and exact repair of systematic parts.

Functional repair has been researched to the greatest extent of the three [14–16]. Dimakis et al. [17] have proposed regenerating codes for functional repairs. They have created a serial theory of the threshold function with informational flow graph. This function describes the relationship between node numbers, surviving nodes numbers, communicating bits, and repair bandwidth. They also studied two extremal points on the optimal trade-off curve between Minimum-Bandwidth Regenerating (MBR) Codes [18] and minimum-storage regenerating (MSR) codes [19, 20]. In MSR codes, the system stores the average bits at each node while ensuring the MDS-code property, so that they are equivalent to standard MDS codes. While in MBR codes, because we are concerned with the minimum bandwidth, MDS codes and non-MDS codes can be used.

After Rashmi et al. [21] had deeply researched MBR and MSR codes, they proposed the model of twin-code framework, which is the trade-off between MBR codes and MSR codes. Under this framework, the nodes are partitioned into two types and encoded using two codes in a manner that reduces the problem of node repair to that of erasure-decoding of the constituent codes. Depending upon the choice of the two codes, the framework can be used to avail one or more of the following advantages: simultaneous minimization of storage space and bandwidth repair, low complexity of operation, fewer disk reads at helper nodes during repair, and error detection and correction.

Exact repair is also proven theoretically. With the idea of interference alignment [22, 23], the exact repair also includes Exact-MBR Codes and Exact-MSR Codes as does functional repair. The idea is to align multiple interference signals in a signal subspace whose dimensions are smaller than the number of interferers.

Different from the theory analysis above, NCCloud has proposed the implementable design for the functional repair. The system has divided each cloud into two parts and uses its algorithm, so that time for data reads can be reduced by 25% compared to time taken to reconstruct the whole file. There are cloud storage systems that provide a scalable platform for storing massive data over multiple storage nodes via erasure coding [24, 25].

3. BMCloud

3.1. Design Goals. In this paper, we want to develop a cloud system applied as a deep archive and make some progress on the base of an existing system. We specifically design BMCloud under a thin cloud assumption—that the remote data center storing the backups does not provide any special backup services, as Cumulus [26] has proposed. The following factors are taken into account.

3.1.1. Low Bandwidth Consumption. Due to the importance of bandwidth consumption, we put it first on this list. Network code has made some advancements on this issue, but we believe that there is still plenty of room for improvement. That is why the bandwidth is one of the most important factors to consider. We wanted to develop an extra layer on functional repair to create a hybrid system, and so finally we chose E Code, which is a RAID-6 code and has excellent I/O properties. In BMCloud, we elaborately apply E Code on the cloud platform and reserve its excellent properties in I/O (see details in Section 3). E Code can also help the system in locating errors and repairing them efficiently. This creation fills in the blanks in the cloud storage system. In BMCloud, cloud nodes are required to have computing abilities, limited to XOR operations, which could easily compute concurrently. However, the response time will not increase due to the reduction in the size of the file's need to transport.

3.1.2. Low Overhead of Maintenance. In BMCloud, we want to improve the functional repair model in some aspects, such as bandwidth and computing cost.

Existing systems such as NCCloud, which use functional repair models, need to regenerate EM to repair failed nodes and that may cause some problems that we should not ignore. Since each fault will lead to whole node data regeneration and EM updates, the maintenance overhead of EM may rise significantly, because it will take many computing resources to generate new EM. After a certain number of faults, the maintenance overhead of the whole system may become unacceptable.

In BMCloud, E Code can deal with faults with size of one or two F-MSR blocks and repair them exactly. Every fault repaired by E Code will not need to regenerate and update EM. This property guarantees that the EM will stay stable for a relatively long time. Furthermore, repairing computing cost of E Code is much less than F-MSR. So the system will not incur cost as high as that for computing and bandwidth resources to maintain stability. Since faults in small scale constitute to the majority of regular faults, E Code can be very helpful.

From that, we know that even though we added an extra layer in our system, there is still room for improvement in terms of better computing and bandwidth performance.

3.1.3. Preservation of Regenerating Code Properties. Though faults on a large scale seldom occur, cloud systems must have a mechanism to deal with these lethal problems. F-MSR code is an excellent approach to a solution and has an acceptable bandwidth cost. We preserve the fault tolerance requirement and repair traffic with F-MSR (with up to a small constant overhead) as compared to the conventional repair method in erasure codes.

In BMCloud, we improve the performance of bandwidth, stability, and other aspects on the premise of the preservation of the properties of F-MSR code.

3.1.4. Flexibility. In BMCloud, we elaborately designed the repair model which made it capable to provide the most economical and stable solutions for faults on different scales. Furthermore, BMCloud has excellent expandability. The number of nodes in E Code can be any number larger than 4, and this property helps the E Code layer connect to F-MSR seamlessly. If a better system using the functional repair model came up, it would be very convenient to deploy E Code on it, so the mechanism of BMCloud can be applied in many situations.

3.2. Notations. For an (n, k)-F-MSR code, we define the $n^{*}(n-k)$ code chunks encoded from $k^{*}(n-k)$ native chunks as F-MSR chunks. Normally, an F-MSR chunk would be placed on a different cloud node. The encode coefficient matrix in

Table 1: Notations of BMCloud.

Notations	Description
F-MSR chunk	A data unit in F-MSR code
Stripe	A coding group containing a collection of strips in E Code
Strip	A data unit of a stripe in E Code
E Code area	Two neighboring cloud nodes in one stripe
Encode matrix (EM)	Coefficient matrix in the F-MSR code
Extended F-MSR chunk	An code chunk after E Code encoding and extension

the F-MSR is defined as the encoding matrix (EM), which is the main part in F-MSR metadata. Each F-MSR chunk will be divided into $(n-1)^{*}n$ strips.

In E Code, we define any two cloud nodes adjacent to each other as an E Code area. An exception occurs for the first cloud node, which is grouped with the last node in an E Code area. To a specific file, an E Code area contains two F-MSR chunks.

A stripe is a concept borrowed from traditional RAID codes, and they are an independent coding group of strips. Any stripe in E Code belongs to specific E Code area.

After E Code coding, an additional strip would be appended to a stripe and more strips in each F-MSR chunk. We define the code chunk after extension as an extended F-MSR chunk. Table 1 shows the notations of BMCloud.

3.3. The Design of BMCloud

3.3.1. Software Architecture of BMCloud. In order to solve the concerns mentioned in Section 3.1, we developed BMCloud. BMCloud is a dual-layer system based on F-MSR shown as Figure 1. The first layer contains the classic F-MSR code, providing the tolerance ability on a cloud level. The second layer implements E Code. Data on every cloud node is firstly encoded by F-MSR code and then encoded by E Code before unloading to the cloud node.

BMCloud consists of three modules as coding, storage, and protocol, and defines four workflows as download, update (upload), delete, and repair. Basic functions in file systems and data structure definitions, as well as consistency control and other utility functions, are also included in the system.

BMCloud is mounted on Linux with FUSE (filesystem in userspace). Basic functions in a filesystem are supported by user-defined codes. Our system implements reading and writing, rename, link, creating a new folder, changing file attributes, and other basic functions only because a perfectly functioned filesystem is not BMCloud's point. BMCloud applies Hadoop zookeeper distributed applications to provide coordination services and ensure the system's strict consistency. In the part of workflows, the system mainly defines the encoding, decoding, update (upload), download, repair, and delete operation workflows. This part is the implementation section about encoding and decoding operation and repair strategy in FMSR and E Code. The underlying part has three modules. Coding module provides a variety of basic encoding methods, including FMSR, RAID0, RAID1, and RS coding, in which we focused on the use of FMSR, a functional repair coding method. Storage module is designed to adapt to different network environments and cloud vendors to provide an interface for basic I/O. Protocol module is a specialized extension module for E Code, defining generation and transmission of the parity blocks in E Code.

3.3.2. E Code Algorithm. Figure 2 is an example of an E Code encoded process. The deep gray squares are the parity strips of E Code. The other 12 strips are data strips which are partitions of F-MSR chunks. In Figure 2, we can see that every data strip belongs to two data links. If n is the number of the nodes, when $n > 3$, E Code can provide double-fault tolerance to protect data in nodes.

Define the row sequence and the column sequence in a strip in the left cloud node as x and y, respectively. So all the left strips in a stripe are equal to $(x+y) \bmod (n+1)$, which is defined as L. When r represents the row sequence in the right cloud node, we see that $L = (2n-1-r) \bmod (n+1)$.

Therefore, we know that

$$L = (x+y) \bmod (n+1) = (2n-1-r) \bmod (n+1). \quad (1)$$

We use an algorithm to improve recovery bandwidth performance and ensure data integrity. The details are given in Algorithm 1.

3.3.3. The Design of Functional Repair. F-MSR code is a code using the functional repair method, which is an important foundation of BMCloud. In F-MSR code, the system utilizes EM to record the mathematical relation between the original data and encoded data. From the EM, the system can retrieve the original data. When faults occur, the system can repair them by calculating a linear transformation on EM and regenerating new encoded data on the cloud nodes.

The encoding process preserves the F-MSR code block. It just adds a parity block of E Code to the data, so that in daily use, there is no need for the system to decode the block twice. The added parity block will only be used in the repair process.

The reason why we choose E Code is that it introduces abundance by adding parity blocks. This method maintains the contents of the data.

The E Code layer can improve the stability of the system while not delaying the response time of the system.

3.3.4. The Design of Exact Repair. The deep gray squars are the parity strips of E Code. The other 12 strips are data strips which are partitions of F-MSR chunks. In Figure 2, we can

```
1: Requirements:
2:     Native chunks;
3:     Encoding request;
4:     Main parameters $(n, k, s)$ in F-MSR code and E Code;
5: Step 1. System initialization
6: Step 2. F-MSR $(n, k)$ encoding
7:     (a): Generate random encoding coefficient vectors;
8:     (b): If MDS is satisfied
9:         then generate an encoding matrix from then coding coefficient vectors;
10:        else return to a;
11:    (c): Compute the product of encoding matrix and native
12:      chunks as F-MSR chunk;
13: Step 3. E Code $(n, s)$ encoding
14:    (a): Divide each F-MSR chunk into stripes and strips;
15:    (b): For stripe A in F-MSR chunks
16:        Calculate the parity strip of this stripe;
17:        Save the parity strip in certain position
18:    (c): Consolidate the strips into extended F-MSR chunks
```

Algorithm 1: Algorithm E Code: encode algorithm of the exact layer.

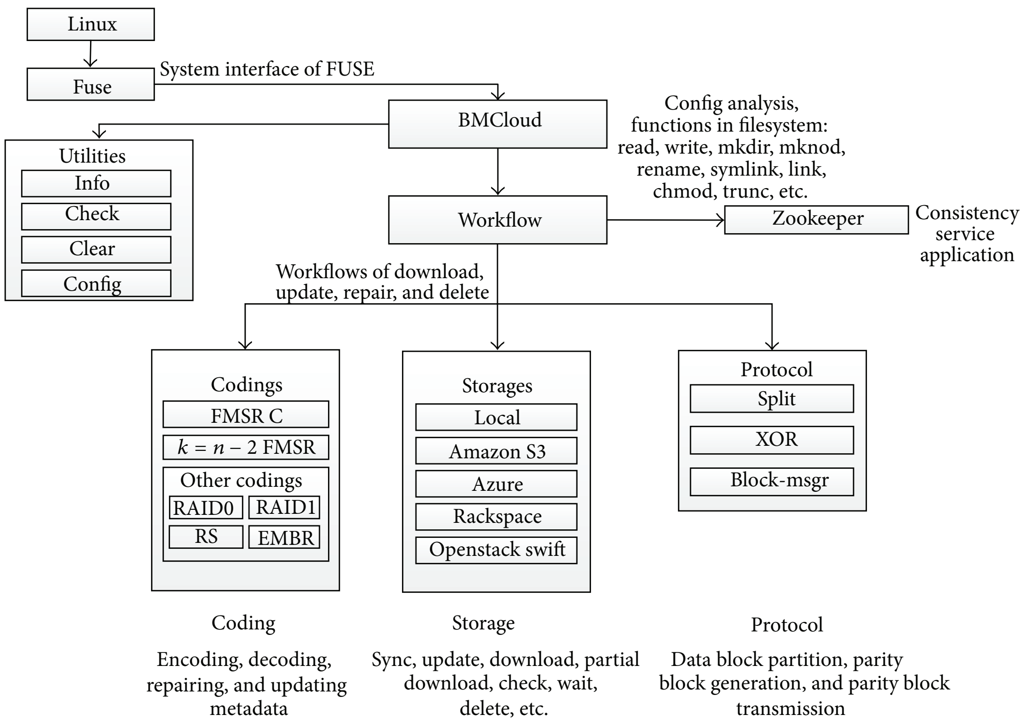
Linux
Fuse
System interface of FUSE
BMCloud
Config analysis, functions in filesystem: read, write, mkdir, mknod, rename, symlink, link, chmod, trunc, etc.
Utilities
Info
Check
Clear
Config
Workflow
Zookeeper
Consistency service application
Workflows of download, update, repair, and delete
Codings
FMSR C
$k = n - 2$ FMSR
Other codings
RAID0
RAID1
RS
EMBR
Storages
Local
Amazon S3
Azure
Rackspace
Openstack swift
Protocol
Split
XOR
Block-msgr
Coding
Encoding, decoding, repairing, and updating metadata
Storage
Sync, update, download, partial download, check, wait, delete, etc.
Protocol
Data block partition, parity block generation, and parity block transmission

Figure 1: Software architecture of BMCloud.

see that every data strip belongs to two data links. If n is the number of the nodes, when $n > 3$, E Code can provide double-fault tolerance to protect data in nodes.

The dividing method provides system repair abilities in smaller granularity. That enables BMCloud to avoid repairing whole nodes in most situations. In other words, BMCloud avoids the cost of calculating the data of a whole node and updating bandwidth costs of EM.

3.3.5. Proxy Architecture. The prototype system of BMCloud is constructed of a proxy and several cloud servers (cloud nodes) in heterogeneous environments. Figure 3 shows a coding group of clouds, which provides storage capacities, redundancy, and computing.

As a controller, a proxy functions to coordinate data transmission between several cloud nodes and to maintain the system's consistency. In the idealized mode, the proxy

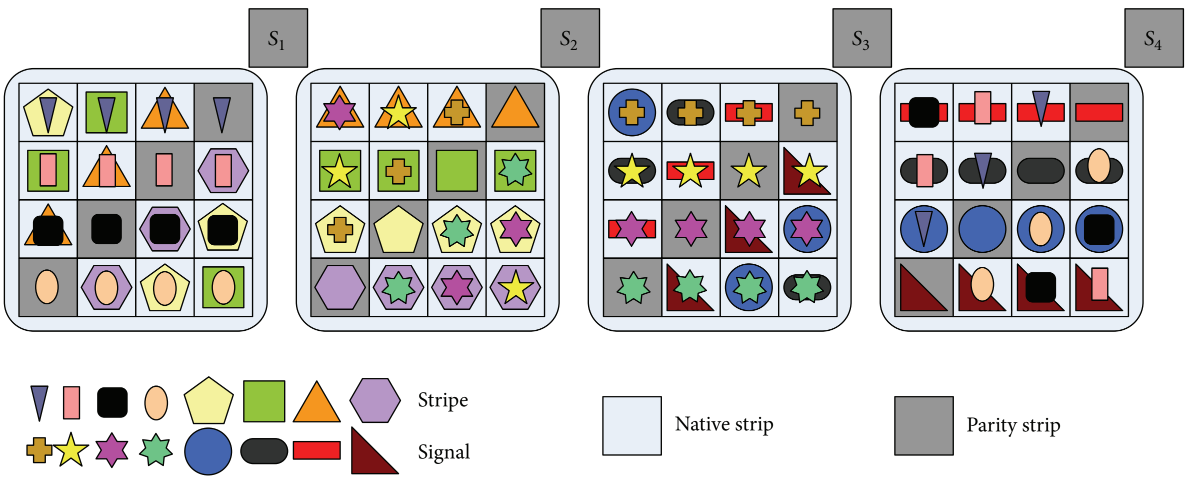

Figure 2: An example of an E Code encoded process.

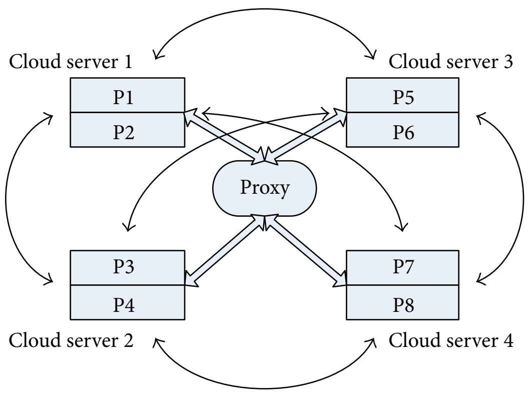

Figure 3: The communication ring of cloud servers.

has no need to compute or store any data, which strengthens the system's scalability. In a large-scale archive environment, proxy process requests remain limited and affordable to current servers or cloud servers, Therefore proxy will not become the bottleneck in this prototype.

Data packages could be directly transmitted between cloud nodes, cutting the bandwidth cost from multiple transmissions. Cloud nodes may need computing capacity to encode and decode the data, undertake the recoveries, and respond to instructions from proxies or requests from other cloud nodes.

We employ a distributed structure in which all the cloud nodes and proxy functions have ability to calculate independently, so that it is necessary to arrange a unified protocol to control and maintain the system.

As with single-fault recovery, there are two equivalent strategies, recovery with the last node or recovery with the next node. In the former case, there will be two kinds of orders from the proxy, the ones to the previous node and the ones to the recovering node. In order to prompt this, we will send an order formatted as follows for prenode (k): $0 \leq k \leq n-2$ or $k = n$. K is the stripe sequence. This order will trigger a repeated XOR operation in the corresponding stripe and the result will be sent to the next node, as the recovery node. Before this order, the proxy should send an order to the recovering node, defined as recnode (x, y). x and y are, respectively, the row and column sequence of the fault. When the recovering node receives this order, it will wait for the coming strips from the previous node. After it is received, the recovering node will operate XOR on the strip and other strips in the row with fault and the result will be the strip needing recovery.

4. The Implement Algorithms

4.1. Encode Algorithms

4.1.1. Encode with F-MSR. Encode the original data with F-MSR code. In this process, the encoded data is divided and distributed to several cloud nodes.

4.1.2. Divide the F-MSR-Encoded Data into E Code Chunks and Add Parity Chunks. Every F-MSR chunk is divided into $(n-1)^{*}n$ strips in this step. The strips will be mapped into certain positions. Strips on the diagonal will be reserved for parity strips in step 3.

4.1.3. Calculate Parity Blocks and Upload Data. Calculate the data of the parity strips and insert it into the reserved space in *the second step*. Figure 2 is an example of a block location. After the calculation, the extended F-MSR chunk will be uploaded to the cloud.

Furthermore, since BMCloud is developed on the foundation of F-MSR code, the system has the cloud level fault tolerance ability. When one cloud node fails, the system can repair the data from the surviving node.

Every neighboring cloud node pair is defined as an E Code area. Any stripe belongs to one and only one E Code Area. In an E Code area, the cloud node on the left owns

$(n-1)$ strips in the stripe, which is leaning towards the top-right corner and the right cloud node has n strips in the stripe showing as a row in Figure 2.

4.2. JUDGE_STYLE Algorithm. In BMCloud, most strips are located on two data-links. In order to improve the recovery ability of BMCloud, we added a parity strip S_n which is the checksum of all the parity blocks on each node (in order to make the Figures more clear, we have hidden S_n in the following figures.). When a fault occurs, the system can search the data link which owns the failed strip and repairs it with the surviving part of the data link. We now take an overview of the related work on the recovery of faults in different scales. We propose the JUDGE_STYLE algorithm, which can judge what kind of style the system will be used. We classify existing recovery solutions into three families, namely exact recovery, enumeration recovery, and functional recovery. Respectively, they are designed for faults of small, medium, and large scales.

4.2.1. Exact Recovery. E Code in our system has excellent recovery abilities. When the number of the failed strips is less than or equal to 4, BMCloud will recover the data with an exact recovery policy. All faults in this scale can be repaired by the E Code layer exactly. In extended-F-MSR chunks, we divide the strips into two families: data strips and parity strips. In these cases, we use vectors (m, n) to represent the situation where there are m failed data strips and n failed parity strips.

(a) (0, 4). Since there are 4 failed parity strips for different data links, which all have only one failed strip, it is obvious that these 4 parity strips can be easily repaired. In the following situations, we will skip all similar relationships.

(b) (1, 3). In this case, there are 3 failed parity strips and 1 failed data strip. If the one failed data strip is in the same data link with the one parity strip, then the other two parity strips can be repaired and the system can repair the rest of the parity strip by using a S1 parity strip. Finally only the data strips can be repaired.

(c) (2, 2). In this situation, we will give an example of every possible approach. We use the symbol $S(m, n)$ to represent the nth strip in the mth cloud.

(i) S(1,3), S(1,6), S(1,10), S(1,13): they can be easily repaired because they are all in a data link, which has only one failed strip.

(ii) S(1,3), S(1,6), S(1,7), S(1,10): S(1,3) and S(1,10) can be repaired first then, the recovery trace of S(1,6) and S(1,7) is obvious.

(iii) S(1,2), S(1,3), S(1,4), S(1,7): S(1,7) first, and from S1 we can repair S(1,4), then S(1,2) can be recovered by a data-link-square, and S(1,3) can by a data-link-upside-down triangle.

(iv) S(1,3), S(1,4), S(1,6), S(1,7): from four involved data-links, we get the equitation set as follows:

$$\begin{aligned} S(1,3)+S(1,4)&=R_1,\\ S(1,3)+S(1,6)&=R_2,\\ S(1,4)+S(1,7)&=R_3,\\ S(1,6)+S(1,7)&=R_4. \end{aligned} \tag{2}$$

R_{1-4} is the result the system calculates from survived strips in the involved data links. So from the equitation set we can calculate the data of the four failed strips.

(d) (3,1) and (4,0). These situations are similar to the situations in (0,4) and (1,3), the repair trace is simple and easy to find so we will not introduce it in detail.

4.2.2. Enumeration Recovery. When the numbers of the failed strips are between 4 and 10. The E Code layer can handle these faults except for in some special situations. So first, BMCloud will scan all the failed strips and check their relationships. Then, the system will try to use a greedy algorithm to repair the fault. The details are in Algorithm 2.

4.2.3. Functional Recovery. When the number of failed strips is larger than 10, the scale of the fault overcomes the upper bound of repair ability of the E Code layer. So BMCloud will repair the fault by the F-MSR layer. The data of the whole node will be regenerated by F-MSR code from the data on surviving cloud servers. After regenerating the new F-MSR chunk, BMCloud will recalculate the parity strips and add them into the F-MSR chunk to restore and extend the F-MSR chunk onto a new cloud sever. In the meanwhile, the related parity strip on the related cloud server will be updated.

5. Evaluation Methodology

5.1. Cost Analyze. Table 2 shows the monthly price plans for three major vendors of cloud storage as of January 2013. We used the price of Azure [27] and assume that the storage usage was within 1 TB/month; data transferred out was more than 1 GB/month but less than 10 TB/month. From the analysis in Section 4, we can save 33.33% of the download traffic during storage repair when $n = 4$. The storage size and the number of chunks being generated per file object of BMCloud are 33.33% larger than RAID-6 when $n = 4$. Since the price of storage is much lower than the bandwidth, the redundancy of BMCloud is acceptable to the user.

However, in the analysis, we have ignored three practical considerations: the computing cost, the size of metadata, and the number of requests issued during repair, because we considered these values negligible in real-life applications.

Computing cost: in real-life applications, the MTTF (99.999999999%) of the Business Cloud is very long, so it will cost few computing resources to guarantee high availability.

Metadata size: in BMCloud, regardless of the size of the data files, the F-MSR metadata size is always within 160 B.

```
1: Requirements:
2:     Native chunks;
3:     Repair request;
4:     Main parameters $(n, k, s)$ in F-MSR code and E Code;
5:     $k$ stands for the number of fail strips
6: Step 1. Basic information collection
7: Step 2. Greedy repair
8:     (a): Scan failed strip $S_i$ $(1 < i < k)$;
9:     (b): If $S_i$can be recovered by a simple data-link
10:         then restore $S_i$, go to Step 1.
11:         else $i = i + 1$;
12:    (c): List the equation set of the rest of the failed strips
13:         from the related data-links
14:    (d): If the equation set is soluble
15:             then restore all the failed strips
16:             else got Step 3.
17: Step 3. Functional Repair
18:    (a): Repair the whole node with F-MSR code;
19:    (b): Recalculate the related parity strip and update all
20:         the EMs on every cloud server;
```

Algorithm 2: Algorithm enumeration recovery.

Table 2: Monthly price plans (in US dollars) for Amazon S3 (US Standard) and Windows Azure Storage, as of January, 2013.

	S3	Azure
Storage (per GB)	\$0.064	\$0.062
Data transfer in (per GB)	free	free
Data transfer out (per GB)	\$0.120	\$0.119
PUT, POST (per 10 K requests)	\$0.100	\$0.010
Get (per 10 K requests)	\$0.010	\$0.010

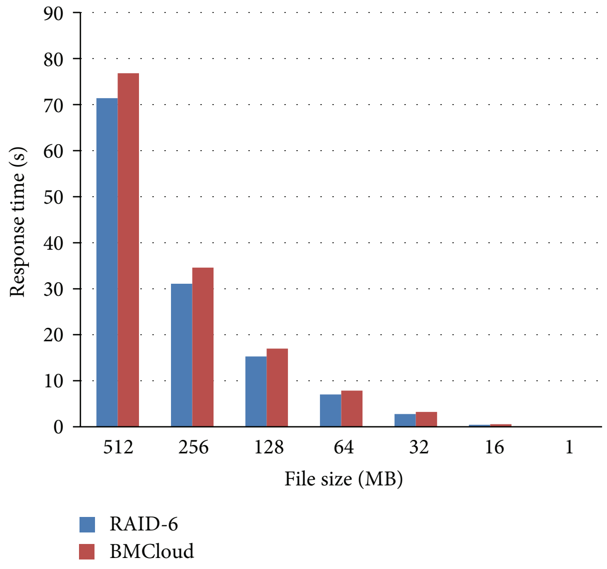

Figure 4: File upload response times of BMCloud on a local cloud.

In evaluation, we used a 512 MB file to test the response time of BMCloud, and compared to the size of the test file, the metadata size will usually negligible. In real-life applications, the size of the data file usually overcomes 1 GB, so the effects caused by metadata on system are too small and do not need evaluation.

Number of Requests. From Table 2, we know that the charge of requests is relatively low compared to storage and bandwidth. RAID-6 and F-MSR differ in the number of requests when recovering data during repairs. Suppose that we store a file of size 4 MB with $n = 4$ and $k = 2$. BMCloud may need special protocol to support this function. When we repair cloud data, there is an agreement that it is merged before the data is transferred over. But if you do not follow these steps, the Get Operation will put all the data transmitted over a result of the flow rate increase.

5.2. Response Time Analysis. In this part, we deploy our BMCloud prototype in a real local cloud environment to evaluate the system performance of the response time. This cloud storage environment is chosen to carry out this analysis in order to evaluate the performance without the effects of network fluctuations. We may continue the analysis on commercial clouds as a future work. All results are averaged over 50 runs.

The experiment is implemented on a storage platform based on OpenStack swift. The proxy of the system is installed on a laptop with Intel Core i5-580 and 8 GB RAM. This machine is connected to an OpenStack swift platform attached to a number of storage servers with Xeon E5606 CPU and 8 GB DDR3 RAM. We create 6 containers on this platform, and 4 containers act as 4 cloud nodes and other 2 containers act as spare nodes, to constitute the testing environment of F-MSR ($n = 4$, $k = 2$) and BMCloud system.

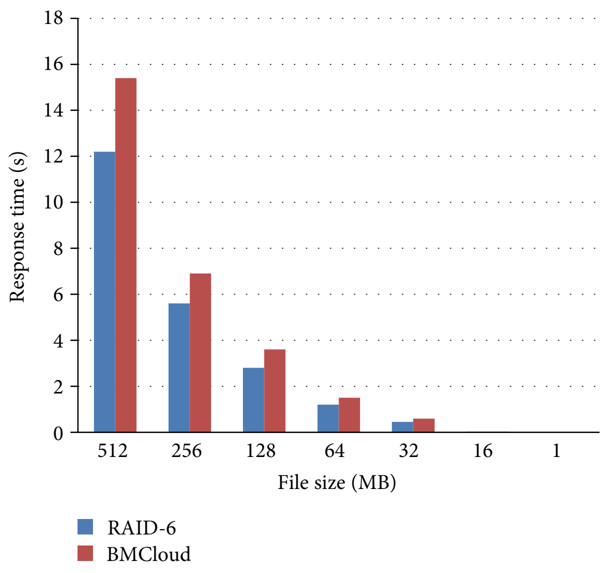

Figure 5: File download response times of BMCloud on a local cloud.

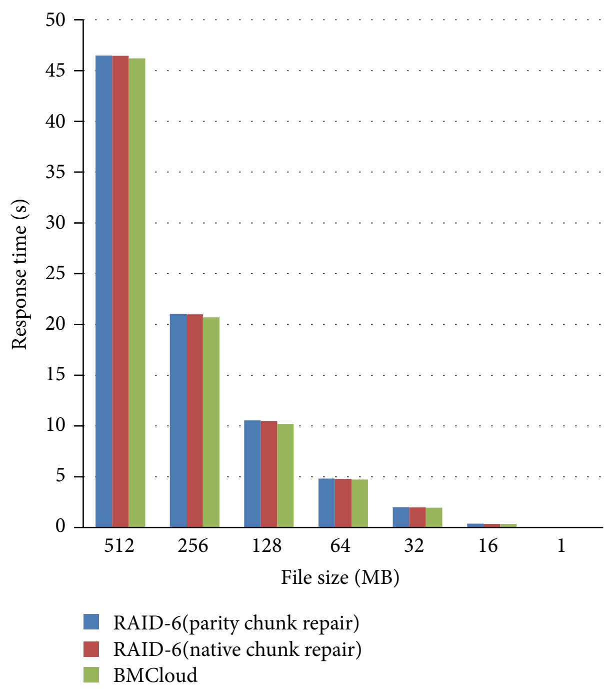

Figure 6: Recovery response times of BMCloud on a local cloud.

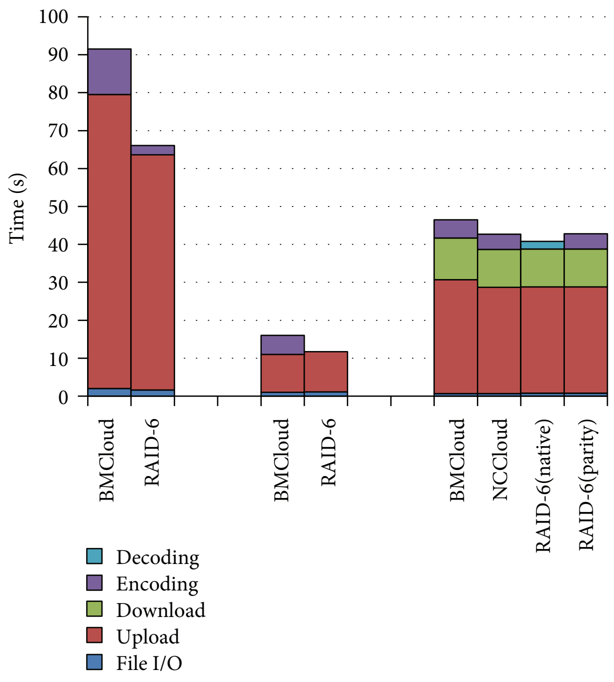

Figure 7: 512 MB file response time detailed analysis (Functional).

We test the response time of the three most important operations in the workflow of BMCloud: file upload, file download, and file recovery. For each operation workflow, we keep a record of the detailed time costs in every type of operation. We use random files sized from 1 MB to 512 MB as the test data set. RAID-6 Reed-Solomon code is chosen as a control group. There are two types of recovery situations, considering whether the failed node is native or parity.

Figures 4, 5 and 6 show response time in 3 main operations workflows, upload, download, and recovery in BMCloud and RAID-6 RS code. BMCloud performs over RAID-6 on response time in file upload and file download.

Figures 7 and 8 show detailed component of the response time in the case of the 512 MB file. We can clearly see that the data transfer time contributes to the main part of the response time for all 3 operation workflows and time costs in both methods are quantitatively similar. In contrast to RAID-6 code, F-MSR code properties make BMCloud present a significant encoding/decoding overhead when a file is uploaded or downloaded. When uploading a 500 MB file, RAID-6 takes 2.496 s to encode and BMCloud takes 12.065 s; when downloading a 500 MB file, BMCloud takes 4.512 s to decode and RAID-6 needs no decoding when native nodes are available. However, uploads and downloads are infrequent operations in an archive storage environment. Moreover, network fluctuation in real environments will balance the difference between RAID-6 and BMCloud.

In the recovery process, BMCloud shows a shorter response time than RAID-6. BMCloud needs to download less data during repairs than RAID-6 and NCCloud. There are two kinds of recovery methods in BMCloud, exact repair, and functional repair. Exact repair sharply curtails repair bandwidth and hence repair response time, showing our main advantage. In repairing a 512 MB file, NCCloud spends 9.593 s in download; the native-chunk repair while RAID-6 spends 12.124 s and BMCloud spends only 5.583 s for all the data. The response time of BMCloud is 41.8% and 53.9% better than that of NCCloud and Raid-6, respectively. On the other hand, BMCloud spends 11.467 s in functional repair, which is a little less than NCCloud and RAID-6.

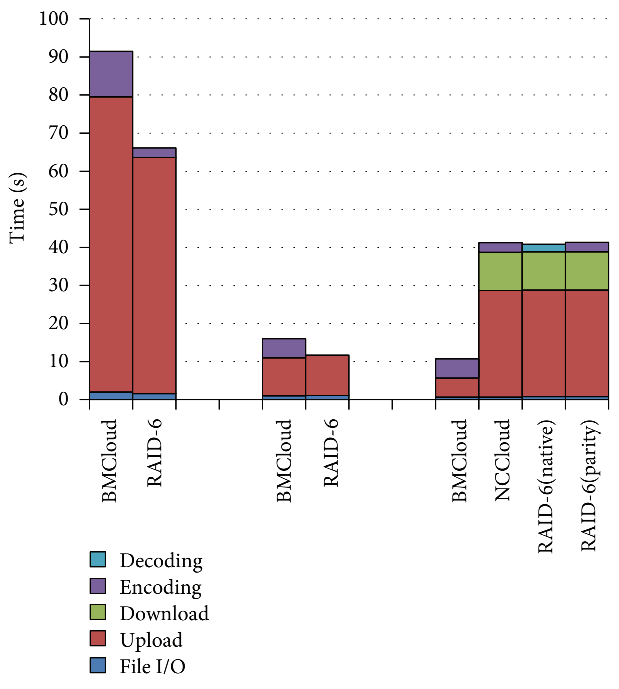

Figure 8: 512 MB file response time detailed analysis (exact).

We have not tested our system on a commercial cloud since our test environment is limited. But from NCCloud [5], we can see that network fluctuation plays a bigger role in determining the response time. So it will be proved that BMCloud will have no performance in functional repair and have good performance when it is in exact repair.

6. Conclusions

In this paper, we developed a low repair bandwidth, low maintenance cost cloud storage system named BMCloud. It has the exact repair algorithm E Code to degrade repair bandwidth and it also provides functional repair to recover data in any condition. The JUDGE_STYLE algorithm can help BMCloud decide in which cases exact repair will be used and in which cases functional repair will be used.

We implemented the system and conducted experiments which prove that BMCloud is effective in degrading repair bandwidth and maintenance costs. The result shows that the response time of BMCloud is 41.8% and 53.9% better than those of NCCloud and Raid-6 when the system operates in the recovery methods of exact repair.

There are still much more work to be done in the future which mainly take two directions. First, we will focus on the recovery of a single-disk failure and do some experiments to verify the performance of BMCloud. Second, we will take a twin-code model into account and make the system more suitable for cloud storage.

In conclusion, we believe that BMCloud is an attractive cloud storage system: one that offers low repair bandwidth, while achieving low maintenance cost.

Acknowledgment

This work is supported in part by the National Basic Research Program of China under Grant no. 2011CB302303, the NSF of China under Grant no. 60933002, and the National High Technology Research and Development Program of China under Grant no. 2013AA013203.

References

[1] Wikipedia. Cloud computing, 2011, http://en.wikipedia.org/wiki/Cloud_computing.

[2] N. Leavitt, "Is cloud computing really ready for prime time?" *Computer*, vol. 42, no. 1, pp. 15–25, 2009.

[3] B. Calder, J. Wang, A. Ogus et al., "Windows azure storage: a highly available cloud storage service with strong consistency," in *Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP '11)*, pp. 143–157, October 2011.

[4] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP '03)*, pp. 29–43, October 2003.

[5] Y. Hu, H. Chen, P. Lee, and Y. Tang, "NCCloud: applying network coding for the storage repair in a cloud-of-clouds," in *Proceedings of the USENIX FAST*, 2012.

[6] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *IEEE Transactions on Information Theory*, vol. 56, no. 9, pp. 4539–4551, 2010.

[7] O. Khan, R. Burns, J. S. Plank, W. Pierce, and C. Huang, "Rethinking erasure codes for cloud file systems: minimizing I/O for recovery and degraded reads," in *Proceedings of the USENIX FAST*, 2012.

[8] D. A. Patterson, P. Chen, G. Gibson, and R. H. Katz, "Introduction to redundant arrays of inexpensive disks (RAID)," in *Proceedings of the ACM SIGMOD International Conference*, pp. 112–117, February 1989.

[9] M. Blaum, J. Brady, J. Bruck, and J. Menon, "Evenodd: an efficient scheme for tolerating double disk failures in raid architectures," *IEEE Transactions on Computers*, vol. 44, no. 2, pp. 192–202, 1995.

[10] P. Corbett, B. English, A. Goel et al., "Row-diagonal parity for double disk failure correction," in *Proceedings of the USENIX FAST*, 2004.

[11] L. Xu and J. Brack, "X-code: MDS array codes with optimal encoding," *IEEE Transactions on Information Theory*, vol. 45, no. 1, pp. 272–276, 1999.

[12] J. Wan, C. Yin, J. Wang, and C. Xie, "A new high-performance, energy-efficient replication storage system with reliability guarantee," in *Proceedings of the 28th Conference on Mass Storage Systems and Technologies*, 2012.

[13] S. Wan, Q. Cao, C. Xie, B. Eckart, and X. He, "Code-M: a non-MDS erasure code scheme to support fast recovery from up to two-disk failures in storage systems," in *Proceedings of the 40th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN '10)*, pp. 51–60, July 2010.

[14] Y. Wu, A. G. Dimakis, and K. Ramchandran, "Deterministic regenerating codes for distributed storage," in *Proceedings of the Allerton Conference on Control, Computing and Communication*, Monticello, Ill, USA, 2007.

[15] Y. Wu, "Existence and construction of capacity-achieving network codes for distributed storage," *IEEE Journal on Selected Areas in Communications*, vol. 28, no. 2, pp. 277–288, 2010.

[16] A. Jiang, "Network coding for joint storage and transmission with minimum cost," in *Proceedings of the IEEE International Symposium on Information Theory (ISIT '06)*, pp. 1359–1363, July 2006.

[17] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *IEEE Transactions on Information Theory*, vol. 56, no. 9, pp. 4539–4551, 2010.

[18] K. V. Rashmi, N. B. Shah, P. V. Kumar, and K. Ramchandran, "Exact regenerating codes for distributed storage," in *Proceedings of the Allerton Conference on Control, Computing and Communication*, Urbana, Ill, USA, 2009.

[19] Y. Wu and A. G. Dimakis, "Reducing repair traffic for erasure coding-based storage via interference alignment," in *Proceedings of the IEEE International Symposium on Information Theory (ISIT '09)*, pp. 2276–2280, Seoul, Republic of Korea, July 2009.

[20] N. B. Shah, K. V. Rashmi, P. V. Kumar, and K. Ramchandran, "Explicit codes minimizing repair bandwidth for distributed storage," in *Proceedings of the IEEE Information Theory Workshop*, 2010.

[21] K. V. Rashmi, N. B. Shah, and P. V. Kumar, "Enabling node repair in any erasure code for distributed storage," in *Proceedings of the Allerton Conference on Control, Computing and Communication*, Urbana, Ill, USA, 2011.

[22] V. R. Cadambe and S. A. Jafar, "Interference alignment and degrees of freedom of the K-user interference channel," *IEEE Transactions on Information Theory*, vol. 54, no. 8, pp. 3425–3441, 2008.

[23] C. Suh and D. Tse, "Interference alignment for cellular networks," in *Proceedings of the 46th Annual Allerton Conference on Communication, Control, and Computing*, pp. 1037–1044, Urbana, Ill, USA, September 2008.

[24] R. Bhagwan, K. Tati, Y. Cheng, S. Savage, and G. Voelker, "Total recall: system support for automated availability management," in *Proceedings of the of NSDI*, 2004.

[25] J. Kubiatowicz, D. Bindel, Y. Chen et al., "OceanStore: an architecture for global-scale persistent storage," in *Proceedings of the 9th Internatinal Conference Architectural Support for Programming Languages and Operating Systems*, pp. 190–201, November 2000.

[26] M. Vrable, S. Savage, and G. Voelker, "Cumulus: Filesystem backup to the cloud," in *Proceedings of the USENIX FAST*, 2009.

[27] G. DeCandia, D. Hastorun, M. Jampani et al., "Dynamo: Amazon's highly available key-value store," in *Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP '07)*, pp. 205–220, October 2007.