Popularity-aware Multi-failure Resilient and Cost-effective Replication for High Data Durability in Cloud Storage

Jinwei Liu, *Member, IEEE, ACM,* Haiying Shen, *Senior Member, IEEE, Member, ACM,* Husnu S. Narman, Zongfang Lin, *Member, IEEE, ACM,* and Zhuozhao Li

Abstract—Large-scale data stores are an increasingly important component of cloud datacenter services. However, cloud storage system usually experiences data loss, hindering data durability. Three-way random replication is commonly used to lead better data durability in cloud storage systems. However, three-way random replication cannot effectively handle correlated machine failures to prevent data loss. Although Copyset Replication and Tiered Replication can reduce data loss in correlated and independent failures, and enhance data durability, they fail to leverage different data popularities to substantially reduce the storage cost and bandwidth cost caused by replication. To address these issues, we present a popularity-aware multi-failure resilient and cost-effective replication (PMCR) scheme for high data durability in cloud storage. PMCR splits the cloud storage system into primary tier and backup tier, and classifies data into hot data, warm data and cold data based on data popularities. To handle both correlated and independent failures, PMCR stores the three replicas of the same data into one Copyset formed by two servers in the primary tier and one server in the backup tier. For the third replicas of warm data and cold data in the backup tier, PMCR uses the compression methods to reduce storage cost and bandwidth cost. Extensive numerical results based on trace parameters and experimental results from real-world Amazon S3 show that PMCR achieves high data durability, low probability of data loss, and low storage cost and bandwidth cost compared to previous replication schemes.

Index Terms—Cloud storage, Replication, Data durability, Cost-effectiveness, SLA.

1 Introduction

Large-scale data stores are an increasingly important component of cloud datacenter services. Cloud providers, such as Amazon S3 [1], Google Cloud Storage (GCS) [2] and Windows Azure [3] offer storage as a service. In the storage as a service, users store their data (i.e., files) into a cloud storage system and retrieve their data from the system. It is critical for cloud providers to reduce Service Level Agreement (SLA) violations to provide high quality of service and reduce the associated penalties for such services. High data durability is usually required by cloud storage systems to meet SLAs. Durability means the data objects that an application has stored into the system are not lost due to machine failures (e.g., disk failure) [4]. For example, services that use Amazon Dynamo storage system typically require that 99.9% of the read and writes requests execute within 300ms [5].

Data loss caused by machine failures typically affects data durability. Machine failures usually can be categorized into correlated machine failures and non-correlated machine failures. Correlated machine failures refer to the events in which multiple nodes (i.e., servers, physical machines) fail concurrently due to the common failure causes [6], [7] (e.g., cluster power outages, workload-triggered software bug manifestations, Denial-of-Service attacks), and this type of failures often occur in large-scale storage systems [8]–[10]. Significant data loss is caused by correlated machine failures [11], [12], which have been documented by Yahoo! [13], LinkedIn [6] and Facebook [14]. Non-correlated machine failures refer to the events in which nodes fail individually (e.g., individual disk failure, kernel crash). Usually, non-correlated machine failures are caused by factors such as different hardware/software compositions and configurations, and varying network access abilities.

The storage demand in a cloud storage system increases exponentially [15]. Data popularity is skewed in cloud storage. The analysis of traces from Yahoo!'s Druid cluster shows that the top 1% of data is an order of magnitude more popular than the bottom 40% [16]. Due to highly skewed data popularity distributions [16], [17], popular data with considerably higher request frequency (referred to as *hot data*) [18] could generate heavy load on some nodes [16], which may result

- *Corresponding Author. Email: hs6ms@virginia.edu; Phone: (434) 924-8271; Fax: (434) 982-2214.*

- *Jinwei Liu is with the Institute for Simulation and Training, University of Central Florida, Orlando, FL 32826, USA. E-mail: jliu@ist.ucf.edu.*
- *Haiying Shen is with the Computer Science Department at the University of Virginia, Charlottesville, VA 22904, USA. E-mail: hs6ms@virginia.edu.*
- *Husnu S. Narman is with the Computer Science Department at Marshall University, Huntington, WV 25755, USA. E-mail: narman@marshall.edu.*
- *Zongfang Lin is with the Huawei US R&D Center, Santa Clara, CA 95050, USA. E-mail: Zongfang.Lin@huawei.com.*
- *Zhuozhao Li is with the Department of Computer Science at University of Chicago, Chicago, IL 60637, USA. E-mail: zhuozhao@uchicago.edu.*

in data unavailability at a time. Availability means that the requested data objects will be able to be returned to users [4]. Actually, much of the data stored in a cloud system is rarely read (commonly referred to as *cold data* [15], [18], [19]). To enhance data availability and durability, data replication is commonly used in cloud storage systems. Replicas of cold data waste the storage resource and generate considerable storage and bandwidth costs (for data updates and data requests) [18] that outweigh their effectiveness on enhancing data durability. Thus, it is important to compress and deduplicate unpopular data objects and store them in low-cost storage medium [15], [20].

Random replication, as a popular replication scheme, has been widely used in cloud storage systems [11], [21]. Cloud storage systems, such as Hadoop Distributed File System (HDFS) [13], RAMCloud [22], Google File System (GFS) [23] and Windows Azure [24] use random replication to replicate their data in three randomly selected servers from different racks to prevent data loss in a single cluster [11], [21], [22], [25]. However, the three-way random replication cannot well handle correlated machine failures because data loss occurs if any combination of three nodes fail simultaneously [11]. To handle correlated machine failures, Copyset Replication [11] and Tiered Replication [21] have been proposed. However, both methods do not try to reduce storage cost or bandwidth cost caused by replication though data replicas bring about considerably high storage and bandwidth costs. Although many replication schemes have been proposed to improve data durability [8], [9], [26]–[29], they do not concurrently consider different data popularities and multiple failures (i.e., correlated and non-correlated machine failures) to increase data availability and durability and reduce the storage and bandwidth costs caused by replication without compromising request delay greatly.

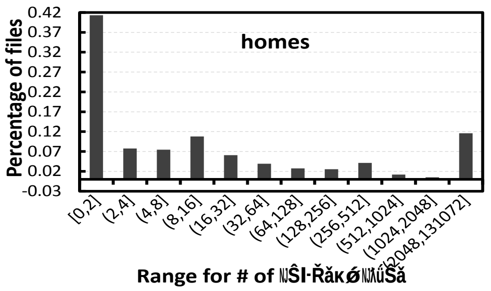
To address the above issues, in this paper, we aim to design a cost-effective replication scheme that can achieve high data durability and availability while reducing storage cost and bandwidth cost caused by replication. To achieve our goal, we propose a popularity-aware multi-failure resilient and cost-effective replication scheme (PMCR), which has advantages over the previous proposed replication schemes because it concurrently owns the following distinguishing features: First, it can handle both correlated and non-correlated machine failures. Second, it compresses rarely used replicas of unpopular data to reduce storage cost and bandwidth cost without compromising the data durability, data availability, and data request delay greatly. We summarize the contributions of this work below.

- We conducted trace data analysis, and the analytical results confirm the existence of read-intensive and write-intensive data, data popularity and data similarity in cloud storage, which lay the solid foundation of the design of PMCR.
- PMCR handles both correlated and independent failures by storing the three replicas of the same data into one Copyset formed by two servers in the primary tier and one server in the backup tier. The primary tier resides close to primary replicas and is used for recovering data with low read latency, and the backup tier is located off-site (e.g., remote location) and serves as the disaster recovery site to protect from site outage or to restore when the local backup is not available.
- PMCR classifies data into hot data, warm data and cold data based on data popularity, and it significantly reduces the storage and bandwidth costs without compromising data durability, data availability, and data request delay greatly by selectively compressing the third replicas of data objects based on data popularity in the backup tier. For read-intensive data, PMCR uses the Similar Compression method (SC), which leverages the similarities among replica chunks and removes redundant replica chunks; for write-intensive data, PMCR uses the Delta Compression method (DC), which records the differences of similar data objects and between sequential data updates.
- Since Balanced Incomplete Block Design (BIBD) does not always exist for any given combination of treatment number, replication level and block size [11], [30], PMCR uses Partially Balanced Incomplete Block Design (PBIBD) to generate the sets of nodes for storing the replicas of the data when the BIBD does not exist, which overcomes the limitation of BIBD and greatly increases the chance of generating the sets of nodes.
- PMCR enhances SC by eliminating the redundant chunks between different data objects (rather than only within one data object) and enhances DC by recording the differences between different data objects (rather than only the difference between sequential updates), and it further reduces the storage and bandwidth costs caused by replication.
- We analyzed the system performance of PMCR in comparison with other replication schemes in terms of storage cost, data durability and bandwidth cost, which shows that PMCR outperforms other schemes in these aspects.
- We have conducted extensive numerical analysis based on trace parameters and experiments on Amazon S3 to compare PMCR with other state-of-the-art replication schemes. Both numerical and experimental results show that PMCR achieves high data durability, low data loss probability and low storage cost and bandwidth cost.

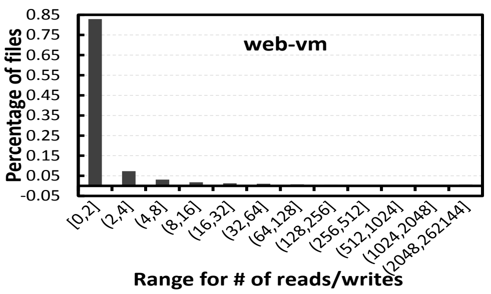
The remainder of this paper is organized as follows. Section 2 presents the analysis of the trace data. Section 3 presents the design for PMCR. Section 4 describes the analysis of system performance. Section 5 presents the numerical and experimental results. Section 6 reviews the related work. Section 7 concludes this paper with remarks on our future work.

2 TRACE DATA ANALYSIS

We collected two real-world traces: a public cloud from CloudVPS [31] and a substantial amount of block I/O traces from a private cloud at Florida International University (FIU). The CloudVPS trace consists of block I/O traces collected from hundreds of VMs on the production system of the IaaS cloud for several days.The FIU trace contains around two months of block I/O traces collected from several production servers (i.e., webserver). It contains the trace for the homes workload, web-vm workload and webserver workload which are for different applications. The homes workload is from a NFS server that serves the home directories of the research group at FIU. The research group activities include software deployment, testing, experimentation, plotting using software and technical document preparation. The web-vm workload is collected from a virtualized system hosting two Computer Science department



(a) Different read/write rates in homes workload



(b) Different read/write rates in web-vm workload

Fig. 1: Percent of files with different number of reads/writes.

web-servers: webmail proxy and online course management system.

2.1 Different Data Popularities

Figure 1(a) shows the different access (including read and write) frequencies of files in the homes workload. We see that around 41% files fall in the range of (0,2], and 10% files fall in the range of (8,16]. The result shows that different files have different access frequencies, and a small percentage of files have very low access frequency or extremely high access frequency. Figure 1(b) shows the different access frequencies of files in the web-vm workload. We see that around 82% files are in the range of (0,2], and 7% files fall in the range of (2,4]. The result also shows that different files have different access frequencies, and a small percentage of files have very low access frequency or extremely high access frequency. Both Figure 1(a) and Figure 1(b) indicate the skewness of popularity distribution of files. Based on this observation, PMCR considers the different popularities of files in file operation, so that the storage and bandwidth costs can be reduced as much as possible without compromising the data durability and availability and file request delay greatly.

2.2 Different Data Intensiveness

To show the existence of write-intensive and read-intensive data in cloud storage system, we measure the FIU webserver trace and CloudVPS trace. Figure 2 shows the number of reads and writes for the FIU webserver trace per week for a total of 35 days. Overall, across the entire trace, there are around 30% of reads and 70% of writes. From this figure, we see that the I/O patterns of FIU webserver are dominated by writes, that is, the FIU webserver data is write-intensive.

Figure 3 shows the number of reads and writes for different VMs in CloudVPS. From the figure, we see that the I/O patterns of some VMs are dominated by reads and the I/O patterns of some VMs are dominated by writes. The results

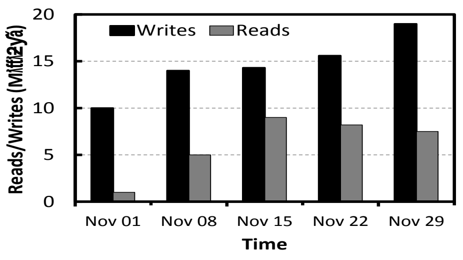


Fig. 2: I/O patterns of the FIU webserver.

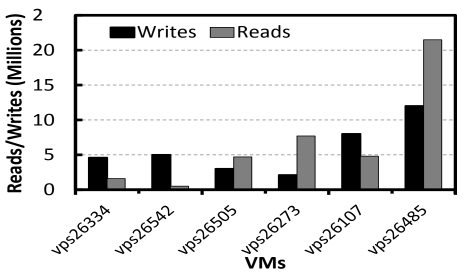


Fig. 3: I/O patterns of the VMs in CloudVPS.

from Figure 2 and Figure 3 confirm the existence of write-intensive and read-intensive data in cloud storage systems. Based on this observation, PMCR uses different compression methods for write-intensive data and read-intensive data in order to reduce the storage and bandwidth costs as much as possible.

2.3 Chunk Similarity

We use two FIU workloads, homes workload and web-vm workload, to analyze the similarity between chunks of data objects. In cloud storage systems, data objects are usually stored in the form of chunks. The chunks usually have some similarity between each other [32], [33]. We grouped the chunks that have no more than 10, 100 and 1000 replicas, respectively, into each group. Then, we calculated the average number of replicas per chunk in each group (called workload similarity). The similarity between two chunks (say A and B) is defined as

$$Sim(A,B) = \frac{|A \cap B|}{A} \tag{1}$$

Figure 4 shows the workload similarity of each group of the homes workload and web-vm workload. From the figure, we see that the workload similarity exists in each group. In the group with no more than 1000 replicas is 8.7 and 4.5 in the homes workload and web-vm workload, respectively. The result shows that data similarity exists among data chunks as indicated in [34]. This observation motivates the design of PMCR, which leverages the similarities between data chunks to eliminate the redundant data chunks in storage and data transmission.

3 System Design

In this section, we first introduce some concepts and assumptions, and then formulate our problem. Finally, we present the design of PMCR based on the observations from the workload analysis.

Suppose there are m data objects and each data object is split into M partitions (i.e., chunks) in the cloud storage system [28], [34]–[36]. A data object is lost if any of its

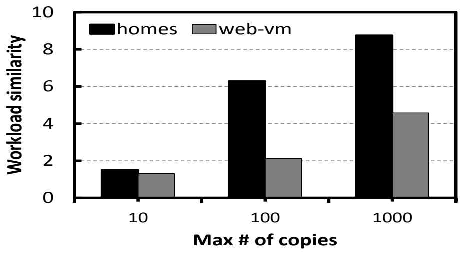


Fig. 4: Data similarity in homes workload and web-vm workload.

partitions is lost [11]. We assume there are N servers in the cloud storage system. For analytical tractability, we assume that a server belongs to a rack, a room, a data-center, a country and a continent. We use the label in the form of "continent-country-datacenter-room-rack-server" to identify the geographic location of a server [28], [37].

Problem Statement: Given data object request probabilities, data object sizes, and failure probability, how to replicate the chunks of data objects so that the node failure probability, storage cost and bandwidth cost are minimized in both correlated failures and non-correlated failures?

To solve this problem, we build a cost-effective replication scheme with data compression to maximize the data durability in both correlated and non-correlated failures while reducing the cost (storage cost and bandwidth cost).

3.1 PMCR Replication Scheme

3.1.1 *Classification of Data Types*

PMCR classifies data into three types: hot data, warm data and cold data based on data popularity. The popularity of a data object is measured by its visit frequency, i.e., the number of visits in a time epoch (denoted by v_i) [17], [28], [38]. That is, $\varphi_i(\cdot) = \alpha \cdot v_i$, where φ_i denotes the popularity of a data object, α is a coefficient. Suppose the time is split into epoches, then the popularity at epoch $t+1$ can be estimated based on the popularity value and coefficient β at epoch t:

$$\varphi_i^{t+1}(\cdot) = \beta \cdot \varphi_i^t(\cdot) + \alpha \cdot v_i \tag{2}$$

To determine the popularity type of a data object, PMCR first calculates the popularity of each data object, and then ranks them based on their popularity values. PMCR considers the data objects with popularity rank within top 25% as hot data, the data objects with popularity rank between $(25\%, 50\%]$ as warm data, and the data objects with popularity rank between $(50\%, 100\%]$ as cold data.

PMCR also needs to determine whether a data object is read-intensive or write-intensive in order to choose a compression method accordingly. For this purpose, it sets thresholds for read rate and write rate. PMCR logs the number of reads and writes of each data object in each time epoch. A data object is write-intensive if its write rate is higher than the pre-defined write rate threshold, and it is read-intensive if its read rate is higher than the pre-defined read rate threshold. PMCR determines the read-intensiveness and write-intensiveness of each data object periodically.

3.1.2 *Replica Placement*

PMCR first splits the nodes in the system into two tiers: primary tier and backup tier. As in the three-way replication, in PMCR, the first two replicas of all data objects are stored in primary tier, and the third replicas of data objects are stored in backup tier. For load balance, the number of nodes in the primary tier is twice of the number of nodes in the backup tier. That is, PMCR assigns $\lfloor \frac{2N}{3} \rfloor$ nodes to the primary tier, and assigns $\lfloor \frac{N}{3} \rfloor$ to the backup tier.

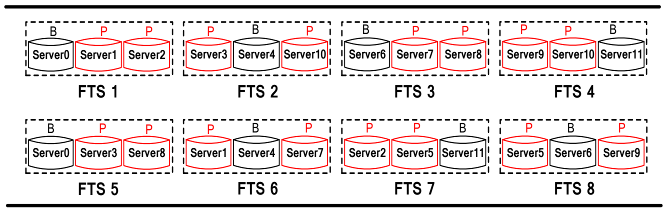


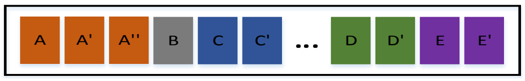
Fig. 5: Fault-tolerant sets (FTSs) in PMCR. (P: Primary tier, B: Backup tier.)

To reduce the data loss caused by correlated machine failures, PMCR adopts the fault-tolerant set (FTS) [11] (i.e., Copyset). An FTS is a distinct set of servers that holds all replicas of a data object's chunk. Each FTS is a single unit of failure because at least one data object is lost when an FTS fails. We will explain the details of FTS in Section 4.2.1. PMCR then partitions the nodes and uses Balanced Incomplete Block Design (BIBD)-based (or Partially Balanced Incomplete Block Design (PBIBD)-based) to generate FTSs. As shown in Figure 5, each FTS contains two nodes from the primary tier and one node from the backup tier, which can protect against correlated machine failures [21]. PMCR replicates each chunk of every data object in a single FTS[1]. For example, in Figure 5, 12 servers are split into two tiers, and there are 8 FTSs across the primary tier and the backup tier. The servers with red lines (marked by "P") are from the primary tier and the servers with black lines (marked by "B") are from the backup tier. PMCR replicates the first two chunk replicas of data objects on the

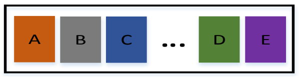
Algorithm 1: Pseudocode for the PMCR algorithm

```
   Input: Data objects' visit frequencies, read and write rates,
          thresholds for determining hot data, warm data and cold
          data
1  Split the nodes (in the system) into primary tier and backup tier
2  Use BIBD-based (or PBIBD-based) method to generate FTSs,
   each FTS contains two nodes from the primary tier and one
   node from the backup tier
3  Compute the popularities of each data object
4  for each data object do
5      if the data is hot data then
6          Store its chunk replicas to the nodes in an FTS, the
           first two chunk replicas are in primary tier and the
           third one is in backup tier
7      else
8          if the data is read-intensive data then
9              Store its chunk replicas to the nodes in an FTS,
               the first two chunk replicas are in primary tier and
               the third one is in backup tier using SC
10         if the data is write-intensive data then
11             Store its chunk replicas to the nodes in an FTS,
               the first two chunk replicas are in primary tier and
               the third one is in backup tier using DC
```

1. Although putting all replicas of a chunk to the nodes in an FTS can bring about the cost of inter-rack transfer (across oversubscribed switches), it can significantly reduce data loss probability caused by correlated machine failures by using BIBD-based method [11].



(a) Grouping similar blocks



(b) Removing redundant copies

Fig. 6: Similar compression.

primary tier, and replicates the third chunk replicas on the backup tier. PMCR compresses the third replicas of warm data and cold data on the backup tier to further reduce the storage cost and bandwidth cost since they are not frequently visited.

Algorithm 1 shows the pseudocode of the PMCR replication algorithm. PMCR splits the nodes in the storage system into primary tier and backup tier (Line 1) [21]. The primary tier stores the first two chunk replicas of data objects and the backup tier stores the third replicas of data objects [21]. This three-way replication and help handle the correlated machine failures. PMCR uses BIBD-based or PBIBD-based method to generate FTS. Each FTS consists of two servers from the primary tier and one server from the backup tier (Line 2). Each chunk will be replicated into one FTS to protect against correlated machine failures. To reduce storage cost and bandwidth cost without compromising data availability of popular data, PMCR classifies data into hot data, warm data and cold data based on popularities of data objects, and determines whether each data object is read-intensive and (or) write-intensive (Line 3), and uses different strategies to store the third replicas of data objects in each data type. Accordingly, PMCR places the replicas of each chunk into the nodes in an FTS (Lines 5-11).

For hot data, PMCR puts the third replicas on the backup tier without compression so that the data can be quickly recovered when the nodes that store the first two replicas fail (Lines 5-6). To further reduce storage cost and bandwidth cost, for warm data and cold data, PMCR puts the third replicas on the backup tier using compression (Lines 8-11). It uses SC to compress read-intensive data (Lines 8-9) and uses DC to compress write-intensive data (Lines 10-11). We will explain the details of SC and DC in Sections 3.2 and 3.3, respectively. The SC method removes the similar chunks within a file or among the files for storage and transmission to the file requester, and the file requester recovers the removed chunks after it receives the compressed file. The DC method stores a copy of a file and the different parts of other files that are similar to this file. For a file request, the stored file copy and the different parts are transmitted to the file requester. In file update, only the updated parts need to be transmitted to the replica nodes. As a result, rather than storing the entire data object, the size of the stored data is greatly reduced with the SC and DC methods. For read-intensive data, rather than transmitting the entire file for a data request, the size of data in transmission is reduced with the SC method. For write-intensive data, rather than transmitting the entire file, only the updated parts are transmitted with the DC method. As a result, the storage and bandwidth costs are greatly reduced. The data recovery for compressed parts may generate a certain delay and overhead when processing a data request. However, the benefits of storage and bandwidth cost saving from the compression outweigh this downside since the warm and cold data objects in the backup tier are rarely read.

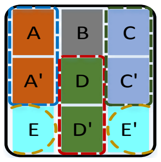


Fig. 7: Intra-file similarity.

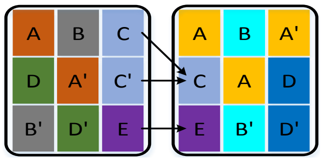


Fig. 8: Inter-file similarity.

3.2 Similar Compression

In SC, similar chunks are grouped together and a certain number of similar chunks form a block. Then, duplicate blocks or near-duplicate blocks to a block are removed. Figure 6 shows an example illustrating the process of grouping similar chunks and compressing the similar chunks together. In Figure 6(a), similar blocks including (A, A', A"), (C, C'), (E, E') are grouped together and they are considered as redundant. In Figure 6(b), for each similar block group, the redundant blocks are removed and only the first block (including A, B, C, D, E) is remained. The data within a data object sometimes are similar to each other [39]. PMCR adopts the SC method to eliminate the redundant chunks within each data object in order to reduce the storage cost and bandwidth cost in data transmission for data requests. Specifically, in PMCR, for read-intensive data objects in the backup tier, for each group of similar blocks, only the first block needs to be stored and all other similar blocks are removed.

Also, PMCR extends the SC method originates from [39] to eliminate the redundant chunks between different data objects to further reduce the costs. We present examples for the intra-file compression and inter-file compression. Figure 7 shows an example of intra-file compression in a file. Similar blocks are marked in the same color. For example, the blocks A and A' are similar blocks; C and C' are similar blocks; D and D' are similar blocks; E and E' are similar blocks. Figure 8 shows an example of inter-file compression. Similar blocks are marked in the same color. The blocks C and C' in the left data object are similar to the block C in the right data object. The block E in the left data object is similar to the block E in the right data object. Similar blocks within a file or between files are grouped together for compression. That is, except the first block, other similar blocks are removed in the storage of a server. An index for a removed block is created to point to the first similar block. When a file requester receives the compressed file, it recovers the removed blocks from the intra-file compression based on the indices. When a received compressed file contains indices pointing to similar blocks in other files caused by inter-file compression, if the file requester has the files, it simply recovers the removed blocks. Otherwise, it requests for these blocks from the cloud to recover the removed blocks. We will explain how to calculate the similarity of blocks in Section 3.4. SC can help reduce the data storage cost due to the reduction of stored data size. It can also reduce the bandwidth cost in responding requested data since removed blocks may not need to transmit.

Algorithm 2 shows the pseudocode of the SC algorithm conducted by each server. Each server first creates chunk blocks with each block containing similar chunks in a file (Line 1). Then, it uses Bloom filter to measure the similarity between

Algorithm 2: Pseudocode for Similar Compression (SC) conducted by each server

```
Input: Data chunks of data objects, threshold for determining
       similarity ($S_{th}$)
1 Create blocks; each block contains similar chunks in a file
2 for each block $blk_s$ do
3     Use Bloom filter to measure the similarity between block
      $blk_s$ and every other block $blk_t$
4     if $BF(blk_s) \cdot BF(blk_t) > S_{th}$ then
          //Dot product of the two Bloom
            filters
5         $blk_s$ and $blk_t$ are considered similar to each other
6         Group $blk_s$ and $blk_t$ together
7     else
8         $blk_s$ and $blk_t$ are considered not similar to each other
9 Use intra-file and inter-file compression for each block group
```

chunk blocks and group similar blocks into a group (Lines 2-8). Specifically, it compares each chunk block with every other chunk block (Line 3). If the two blocks are similar to each other, SC groups the blocks together (Lines 4-6). Finally, the server compresses the similar chunk blocks grouped together (Line 9).

3.3 Delta Compression

Write-intensive data objects have frequent updates. To reduce the cost caused by replication, PMCR uses Delta Compression (DC) to compress the third replicas of the data objects in the backup tier. Figure 9 uses an example to illustrate the process of DC. In Figure 9, chunk B and chunk B' are similar chunks. The regions of difference between chunk B and chunk B' are marked in orange. DC stores chunk B and the differences for chunk B'. When chunk B or chunk B' is updated, only the updated parts rather than the entire chunk are sent to the replica servers. Then, the replica servers update the corresponding parts accordingly. To send a chunk to a file requester, the stored different parts of this chunk and the other parts from the stored entire chunk (chunk B in the above example) are transmitted. Since duplicated parts are removed in the storage, the storage cost is reduced. Also, the bandwidth cost for data updates and for data responses is reduced. We will explain how to calculate the similarity of chunks in Section 3.4.

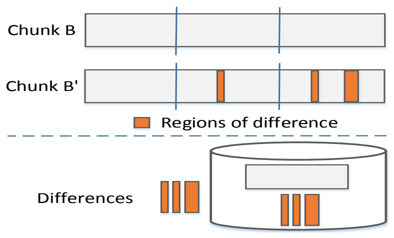


Fig. 9: Delta compression.

A user in the cloud storage system sends a read request for a data object. For each chunk of the data object, PMCR first checks if it is in the primary tier. If it is in the primary tier, PMCR chooses the replica of the chunk from the node with a shorter geographic distance to the user and returns the chunk to the user. Otherwise, PMCR fetches the replica from the nodes located in the backup tier with shorter geographic distance to the user, and sends it to the user. If the data object is warm data

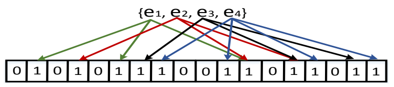


Fig. 10: An example of the Bloom filter of the set $\{e_1, e_2, e_3, e_4\}$.

or cold data, PMCR sends the compressed data object to the user, then the data object will be decompressed on the client-side. We will explain the method to measure the geographic distance between servers in Section 3.5.

When a user sends a write request for a data object, PMCR first checks the popularity type of the data object. If the data object is hot data object, PMCR updates the first two replicas and the third replica without compression. Otherwise, the data object is warm or cold data object. Then, PMCR further checks if the data object is read-intensive or write-intensive. If the data object is read-intensive, PMCR uses SC to compress the third replica in the backup tier. If the data object is write-intensive, PMCR uses DC to compression the third replica.

3.4 Similarity Calculation

To remove the redundant replicas of the data chunks in the backup tier, first we need to find the duplicate (identical) or similar replicas. In this paper, we use the Bloom filter technique to detect the similarity between data blocks or chunks. Compared to other similarity detection methods, Bloom filter enables fast comparison as matching is a simple bitwise-AND operation and generates lower computing overhead. Also, the chunks can be uniquely identified by the SHA-1 hash signature, also called fingerprint. As the amount of data increases, more fingerprints need to be generated, which consume more storage space and incur more time overhead for index searching. To overcome the scalability of fingerprint-index search, PMCR groups a certain number of chunks into a block, and detects the similarity between blocks. Below, we introduce the Bloom filter for detecting similarity between data blocks and will extend this algorithm for detecting similarity between data chunks.

Denote $\mathscr{E} = \{e_1, ..., e_{|\mathscr{E}|}\}$ as a set of chunks of a block. As shown in Figure 10, the Bloom filter for each set $\mathscr{E}$ is represented as a bit array of u bits, with all bits initialized to 0 [40]. Each element e $(e \in \mathscr{E})$ is hashed using k different hash functions, i.e., $h_1, ..., h_k$. The hash functions return values between 1 and u and each hash function maps e to one of the u array positions with a uniform random distribution. To add an element to the set, the Bloom filter feeds it to each of the k hash functions to get k array positions, and sets the k bits corresponding to the hash functions' output, in the Bloom filter to 1. If a bit has already been set to 1, it stays 1. Figure 10 shows an example of the Bloom filter of the set $\{e_1, e_2, e_3, e_4\}$ with $u = 18$ and $k = 3$. The colored arrows indicate the positions in the 18 bit array that each set element is hashed to.

The chunks of a block is a set in Bloom filter parlance whose elements are the chunks. Data blocks that are similar to each other have a large number of common 1s among their Bloom filters. To find similar blocks of a given block, we compare the Bloom filter of the block with the Bloom filter of all the other blocks. The blocks that have the percentage of common 1s higher than a certain threshold (e.g., 70%) are considered as similar blocks [33]. For example, data block A has $\{0, 1, 1, 0, 1, 1, 1, 1, 0, 1\}$ as Bloom

Filter array for its A_1, A_2, *and* A_3 chunks. Data block B has $\{0,1,1,0,1,1,1,0,1,1\}$ as Bloom Filter array for its B_1, B_2, *and* B_3 chunks. If threshold is 70%, then A and B are similar blocks. If the threshold is 100%, then A and B are not similar blocks. To detect similar chunks, we can consider a block as a chunk and consider a chunk as a sub-chunk in the above algorithm and use the same algorithm.

3.5 Distance Calculation

We adopt the method in [28] to compute the geographic distance between servers. The method uses a 6-bit number to represent the locations of servers. Each bit corresponds to the location part of a server, i.e., continent-country-datacenter-room-rack-server. To calculate the distance difference between two servers, starting with the most significant bit, each location part of both servers are compared one by one to compute the geo-similarity between them. If the location parts are equivalent, the corresponding bit is set to 1. Otherwise, the corresponding bit is set to 0. Once a bit is set to 0, all of its lower significant bits are automatically set to 0. For example, given two particular and arbitrary servers S_i and S_j. If the distance between them is represented as 111000 (as shown in below), it indicates that S_i and S_j are in the same datacenter but not in the same room.

continent	*country*	*datacenter*	*room*	*rack*	*server*
1	1	1	0	0	0

The geographic distance is obtained by applying a binary "NOT" operation to the geo-similarity. In this example, it is

$$\overline{111000} = 000111 = 7\ (decimal)$$

4 Analysis of System Performance

4.1 Storage Cost Reduction

Different storage mediums have different costs per unit size. For example, SSD is more expensive than disk, and disk is more expensive than tape. To reduce the storage cost while satisfying the SLA requirements of different applications, we need to decide the storage mediums for different data objects in different tiers (i.e., primary tier and backup tier). The primary tier stores the data objects' first two replicas that are for data availability, and the backup tier stores the data objects' third replicas and it is mainly used to enhance durability. The first two replicas of the data objects in the primary tier are always used for failure recovery, and the third replica is used for failure recovery only if the first two replicas in the primary tier fail simultaneously. Thus, the replicas in the backup tier have lower read frequency compared to the replicas in the primary tier. Therefore, the replicas in the backup tier can be stored on cheaper storage mediums (e.g., tape, disk), and the replicas in the primary tier can be stored on relatively fast and expensive storage mediums (e.g., Memory, SSD). Hot data with considerably higher request frequency could generate heavy load on some nodes, which may lead to data unavailability at a time, and cold data with lower request frequency may waste the storage resource and increase the storage cost. Thus, it is important to choose the storage mediums for storing data based on the popularities of data objects.

To reduce the storage cost (as shown in Figure 11), we choose SSD to store the first two replicas of a hot data object

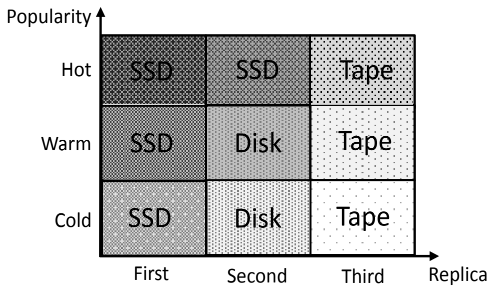


Fig. 11: Selecting storage mediums for data objects' replicas based on their popularities and the tiers where they are located.

and choose tape to store its third replica; we choose SSD to store the first replica of warm data and cold data, and choose disk to store their second replica, and choose tape to store their third replica with compression.

In the following, we analyze the performance of storage cost saving of PMCR. Denote s_i as the size of data object d_i without compression. Define I_c as an indicator function representing whether the third replica of a data object needs to be compressed. Given a data object d_i, we have

$$I_c(d_i) = \begin{cases} 1, & if\ data\ object\ d_i\ is\ hot\ data \\ 0, & if\ data\ object\ d_i\ is\ warm\ data\ or\ cold\ data \end{cases} \tag{3}$$

To represent the actual storage consumption of a data object, we define an indicator function I_s:

$$I_s(d_i) = \begin{cases} 1, & if\ data\ object\ d_i\ is\ compressed \\ 0, & if\ data\ object\ d_i\ is\ not\ compressed \end{cases} \tag{4}$$

Hence, the storage consumption of data object d_i with compression can be calculated as follows:

$$s_i' = I_s(d_i) \cdot \frac{s_i}{\gamma} + (1 - I_s(d_i)) \cdot s_i \tag{5}$$

where γ is the compression ratio, which is defined as the ratio between the uncompressed size and compressed size. The total storage consumption for three-way replication is

$$O_s = \sum_{i=1}^{m} (2 \cdot s_i + I_s(d_i) \cdot s_i' + (1 - I_s(d_i)) \cdot s_i) \tag{6}$$

where m is the number of data objects in the cloud storage system.

Denote c_i ($i \in \{1,2,3\}$) as the unit cost of SSD, disk and tape, respectively. The total storage cost (denoted by C_s) of PMCR is

$$\begin{aligned} C_s = \sum_{i=1}^{m} ((c_1 + (c_1 I_c(d_i) + c_2 \cdot (1 - I_c(d_i))))s_i + c_3(I_c(d_i)s_i \\ + (1 - I_c(d_i))s_i')) \end{aligned} \tag{7}$$

where s_i' is the storage consumption of data object d_i with compression. Compared to previous replication schemes with the consideration of data popularity but without compression [9], PMCR obtains the following storage cost savings:

$$C_s^s = \sum_{i=1}^{m} (c_1 + (c_1 I_c(d_i)) + c_2 \cdot (1 - I_c(d_i)) + c_3)s_i - C_s \tag{8}$$

Compared to the replication schemes without compression or the consideration of data popularity (assume all replicas are stored on SSD for fast recovery), PMCR obtains the following storage cost savings:

$$C_s^s = \sum_{i=1}^{m} (c_1 s_i) - C_s \tag{9}$$

4.2 Data Durability Enhancement

4.2.1 Correlated Machine Failures

Recall that PMCR adopts FTS [11] to correlated machine failures. Each FTS is a single unit of failure because at least one data object is lost when an FTS fails. As the number of FTS increases, the probability of data loss caused by correlated machine failures increases because the probability that the failed servers constitute at least one FTS increases. Hence, the probability data loss caused by correlated machine failures can be minimized by minimizing the number of FTSs.

The probability of failure in correlated machine failures is equal to the ratio of the number of FTSs over the maximum possible number of sets.

$$\frac{\#FTSs}{max\{\#sets\}} \tag{10}$$

Based on the work [11], the probability of failure in correlated machine failures is

$$p_{cor} = \frac{S}{R-1}\frac{N}{R}/\binom{N}{R} \tag{11}$$

where S denotes the scatter width (the number of servers that could be used to store the secondary replicas of a chunk), R denotes the size of FTS (i.e., the number of servers in one FTS). Based on the work [11], the probability of failure in correlated machine failures in random replication can be obtained by substituting "$\#FTSs$" in Formula (10) by the number of FTSs created in random replication.

The following example illustrates the process of generating FTSs. Suppose a storage system has $N = 12$ servers, the size of FTS $R = 3$, and $S = 4$. Using the BIBD-based method, one solution for achieving less number of FTSs is as follows:
$B_1 = \{0,1,2\}, B_2 = \{3,4,10\}, B_3 = \{6,7,8\}, B_4 = \{9,10,11\}$,
$B_5 = \{0,3,8\}, B_6 = \{1,4,7\}, B_7 = \{2,5,11\}, B_8 = \{5,6,9\}$.

The number of FTSs is 8. Therefore, the probability of data loss caused by correlated machine failures is:

$$\#FTSs/\binom{N}{R} = 8/\binom{12}{3} = 0.036$$

However, the number of FTSs in random replication is 72. Hence, the probability of data loss caused by correlated machine failure in random replication is:

$$\#FTSs/\binom{N}{R} = 72/\binom{12}{3} = 0.327$$

There are many methods for constructing BIBDs, but no single method can create optimal BIBDs for any given combination of N and R [41], [42]. Copyset Replication combines BIBD and random replication to generate a non-optimal design. When BIBD-based method cannot find BIBD, PBIBD can be used to generate the sets of nodes for storing the replicas of the data. PBIBD overcomes the limitation of BIBD and greatly increases the chance of generating the sets of nodes. Although the PBIBD is not an optimal approach, it can increase the probability of successfully generating the FTSs for the given combination.

4.2.2 Non-correlated Machine Failures

In non-correlated machine failures, the failure events of machines are statistically independent of each other. They can be categorized into uniform and nonuniform machine failures. In the scenario of uniform machine failures, each machine fails with the same probability, denoted by p ($0 < p < 1$), possibly due to the same computer configuration. The data object is lost if any chunk of the data object is lost, and a chunk is lost only if all the replicas of the chunk are lost. In this analysis, we assume each data object has three replicas. Hence, a chunk loss probability is $p_{uni} = p^3$, and the expected number of chunk loss per data object due to uniform machine failure is

$$E_{p_{uni}} = (\sum_{j=1}^{m} M \cdot p^3)/m \tag{12}$$

where M is the number of chunks for each data object, and m is the number of data objects.

In the scenario of nonuniform machine failures, each machine fails with different probabilities, denoted by p_i ($0 < p_i < 1$), possibly due to different hardware/software compositions and configurations. We assume replicas of data objects are placed on machines with no concern for individual machine failures. Denote $p_1, ..., p_N$ as the failure probabilities of N servers in the cloud storage system, respectively. According to the work [9], the expected data object failure probability is the same as that on uniform failure machines with per-machine failure probability equaling $\sum_{i=1}^{N} p_i/N$. Hence, an approximation of chunk loss probability is $p_{non} = (\sum_{i=1}^{N} p_i/N)^3$ (the actual data chunk loss probability for certain machines would be $p_i * p_j * p_k$ where i, j and k represent the machines which store the data chunks and p_i, p_j, p_k are the failure probobalities of those machines). The approximate number of expected data chunk loss per data object caused by nonuniform machine failure is

$$E_{p_{non}} = (\sum_{j=1}^{m} M \cdot (\sum_{k=1}^{N} p_k/N)^3)/m \tag{13}$$

4.2.3 Correlated and Non-correlated Machine Failures

Denote F as the event that failure occurs, U_1 as the event that correlated machine failures occur, U_2 as the event that uniform machine failure occurs, and U_3 as the event that nonuniform machine failure occurs. Based on previous works [11], [29], both correlated and non-correlated machine failures (uniform and nonuniform machine failures) exist in cloud storage system, and any type of machine failures can incur data loss. Then, the probability of data loss caused by machine failures (correlated and non-correlated machine failures) is obtained as follows

$$P(F) = \sum_{i=1}^{3} P(F|U_i)P(U_i) \quad (\sum_{i=1}^{3} P(U_i) = 1) \tag{14}$$

where $P(F|U_1)$ (p_{cor} in Formula (11)), $P(F|U_2)$ (p_{uni} in Formula (12)) and $P(F|U_3)$ (p_{non} in Formula (13)) are the probabilities of a data object loss due to correlated machine failures, uniform machine failure and nonuniform machine failure, respectively. $P(U_1)$, $P(U_2)$, and $P(U_3)$ are the probabilities of the occurrences of correlated machine failures, uniform machine failure and nonuniform machine failure, respectively.

4.3 Bandwidth Cost Reduction

Replication can enhance data durability and availability but may incur bandwidth cost because the bandwidth is required to keep replicas synchronized [43]. To reduce bandwidth cost, PMCR first categorizes data into read-intensive and write-intensive data based on the historical operations (i.e., read and write) on the data [44]. Then PMCR uses SC to compress the third replicas of read-intensive data, and uses DC to compress the third replicas of write-intensive data.

Based on the previous work [45], the data object write overhead is linear with the number of data object replicas. The bandwidth cost of a data object's partition caused by maintaining the consistency between the replicas of the partition can be approximated as the product of the number of replicas of the partition and the average communication cost parameter (denoted by μ_{com}) [28], i.e., $3\mu_{com}$ (for three-way replication). Thus, the total bandwidth cost of all data objects caused by maintaining the consistency between the replicas of data objects can be calculated as follows:

$$C_b^c = \sum_{j=1}^{m}(3 \cdot M \cdot \mu_{com}) \tag{15}$$

Based on the work [46], the fixed average communication cost can be computed as follows:

$$\mu_{com} = s^u E[\sum_{i,j} dis(S_i, S_j) \cdot \sigma] \tag{16}$$

where s^u is the average update message size, $dis(S_i, S_j)$ is the geographic distance between the server storing the original copy S_i (referred to as *primary server*) and a replica server S_j. The geographic distance is an expectation of all possible distances between primary server and replica servers, which is calculated from a probabilistic perspective. σ is the average communication cost of a unit data per unit distance.

A storage system should be capable of recovering from the loss of data when failures occur, which preserves the reliability guarantees of the system over time. Failure recovery also results in bandwidth cost. When a node fails, all data chunks it was hosting need to be recreated on a new node (We assume a new node is available for replacing the faulty ones [18].), that is, a new node needs to download the data stored on the faulty node to repair the data and replace the failure node.

For simplicity, we assume the data in primary tier and backup tier is evenly distributed over the servers. Hence, the total bandwidth cost caused by recovering data, denoted by C_b^r, is

$$\begin{aligned} C_b^r =& (\frac{\sum_{i=1}^{m}(2 \cdot s_i)}{\lfloor \frac{2N}{3} \rfloor} \lceil \frac{2NP(F)}{3} \rceil \\ &+ \frac{\sum_{i=1}^{m}(I_s(d_i)s_i' + (1 - I_s(d_i))s_i)}{\lfloor \frac{N}{3} \rfloor} \lfloor \frac{NP(F)}{3} \rfloor) \cdot \delta_d \end{aligned} \tag{17}$$

where s_i' and s_i are the size of the data object d_i with and without compression, respectively, and N is the number of nodes in the cloud storage system. $\sum_{i=1}^{m}(2 \cdot s_i)/\lfloor \frac{2N}{3} \rfloor$ and $\sum_{i=1}^{m}(I_s(d_i)s_i' + (1 - I_s(d_i))s_i)/\lfloor \frac{N}{3} \rfloor$ are the average amount of data stored on a server in the primary tier and the backup tier for three-way replication, respectively. $\lceil \frac{2NP(F)}{3} \rceil$ and $\lfloor \frac{NP(F)}{3} \rfloor$ are the number of failure nodes in the primary tier and the backup tier, respectively. δ_d is the average communication cost per unit of data between primary servers and replica servers in the storage system, and it is calculated as $E[\sum_{i,j} dis(S_i, S_j) \cdot \sigma]$.

Based on Formulas (15) and (17), the total bandwidth cost caused by consistency maintenance and data recovery is

$$C_b = C_b^c + C_b^r \tag{18}$$

TABLE 1: Parameters from publicly available data [11].

System	Chunks per node	Cluster size	Scatter width
Facebook	10000	1000-5000	10
HDFS	10000	100-10000	200

TABLE 2: Parameter settings.

Parameter	Meaning	Setting
N	# of servers	1000-10000
M	# of chunks of a data object	50 [47]
R	# of servers in each FTS	3
λ	# of FTSs containing a pair of servers	1
S	Scatter width	4
p	Prob. of a server failure	0.5
m	# of data objects	10000-50000

Compared to previous replication schemes without compression [11], the bandwidth cost savings obtained by PMCR is around:

$$C_b^s = (\frac{\sum_{i=1}^{m} s_i}{\lfloor \frac{N}{3} \rfloor} - \frac{\sum_{i=1}^{m}(I_s(d_i)s_i' + (1 - I_s(d_i))s_i)}{\lfloor \frac{N}{3} \rfloor}).\delta_d \tag{19}$$

5 Performance Evaluation

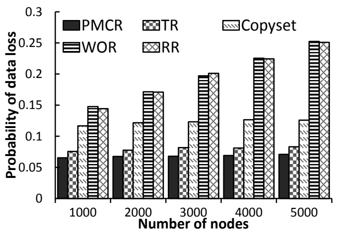
We conducted the numerical analysis based on the parameters in [11] (Table 2) derived from the system statistics from Facebook and HDFS [6], [11], [13], [14], [22], [48], and also conducted real-world experiments on Amazon S3.

5.1 Numerical Analysis

We conducted numerical analysis under various scenarios. We compare our method with the other replication schemes: Random Replication (RR), Copyset Replication (Copyset) [11], Tiered Replication (TR) [21] and WAN Optimized Replication (WOR) [49]. RR is based on Facebook's design, which chooses secondary replica holders from a window of nodes around the primary node. We use R to denote the number of replicas for each data chunk. Specifically, RR places the primary replica on a random node (say node i) in the system, and places the secondary replicas on $(R-1)$ nodes around the primary node (i.e., nodes i+1, i+2,...)[2]. Copyset splits the nodes into a number of Copysets, and constrains the replicas of every chunk to a single Copyset so that it can reduce the frequency of data loss by minimizing the number of Copysets for correlated machine failures. TR stores the first two replicas of a data object in the primary tier for protecting against independent node failures, and stores the third replica in the backup tier for protecting against correlated failures. WOR uses three-way random replication and Delta Compression for replication of backup datasets. The storage medium for the third replica is disk in RR, Copyset and WOR, and is disk or tape that is randomly chosen in TR. The number of nodes that experience concurrent failures in the system was set to 1% of the nodes in the system [50]. We randomly generated 6 bit number from reasonable ranges for each node to represent its location. The distributions of the file popularity and updates follow those of FIU trace. Table 2 shows the parameter settings in our analysis unless otherwise specified.

We first calculate the probability of data loss for each method. We use Formula (14) to calculate the probability of

2. RR is based on Facebook's design, which chooses secondary replica holders from a window of nodes around the primary node.


(a) Facebook

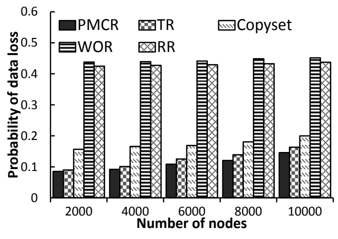
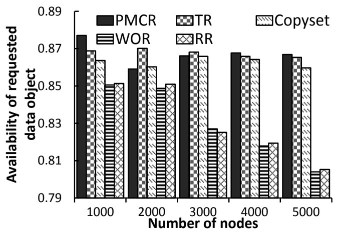

(b) HDFS

Fig. 12: Probability of data loss vs. the number of nodes.


(a) Facebook

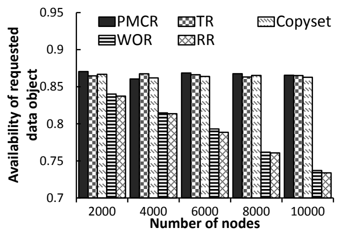
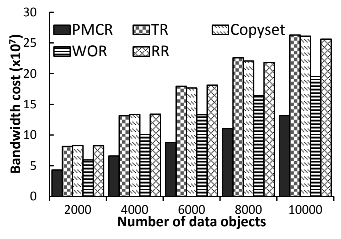

(b) HDFS

Fig. 13: Availability of requested data object vs. the number of nodes.

data loss for PMCR and Formula (11) for Copyset. We use the method in [11] to calculate the data loss probability of random replication for RR and WOR, and use the method in [21] to calculate the data loss probability for TR. Figure 12(a) and Figure 12(b) show the relationship between the probability of data loss and the number of nodes in the Facebook and HDFS environments, respectively. We see that the probability of data loss follows PMCR<TR<Copyset<RR≈WOR. PMCR, TR and Copyset generate lower probabilities of data loss than RR and WOR because they constrain the replicas of a data object to an FTS which can reduce the probability of data loss in correlated machine failures. TR and PMCR generate lower probabilities of data loss than Copyset because they separate the primary data from the backup data by storing the backup data on a remote site, which can further reduce the correlation in failures between nodes in the primary tier and the backup tier [21]. The probability of data loss in PMCR is slightly lower than TR because PMCR chooses different storage mediums for data with different popularities, which decreases the probability of the occurrence of correlated machine failures.

We then calculate the availability of request data object by $1-\sum_{i=1}^{m} r_i M\cdot(P(F))^3$, where r_i is the normalized probability of requesting data d_i. Figure 13(a) and Figure 13(b) show the relationship between the availability of requested data objects and the number of nodes in the Facebook and HDFS environments, respectively. We observe that the availability follows PMCR>TR>Copyset>RR≈WOR. PMCR, TR and Copyset produce greater data availability than RR and WOR because they constrain the replicas of a data object to an FTS to reduce the probability of data loss caused by correlated machine failures and thus increase the availability of data object requests. PMCR and TR generate higher data availability than Copyset because they separate the primary data from the backup data by storing the backup data on a remote site, which can further reduce the correlation in failures between nodes in the primary tier and the backup tier [21]. Therefore, PMCR and TR have higher availability of requested data objects than Copyset.


(a) Facebook

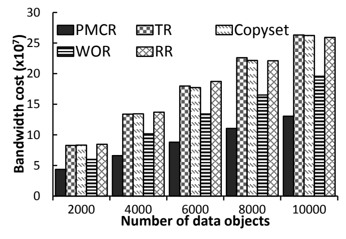
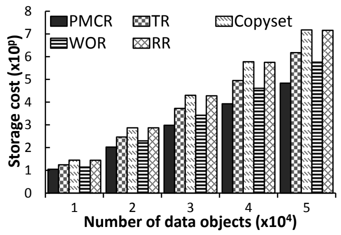

(b) HDFS

Fig. 14: Bandwidth cost vs. the number of data objects.


(a) Facebook

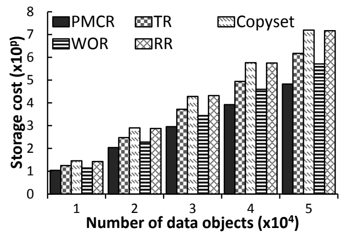
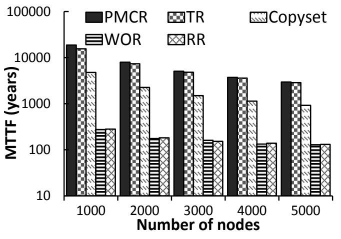

(b) HDFS

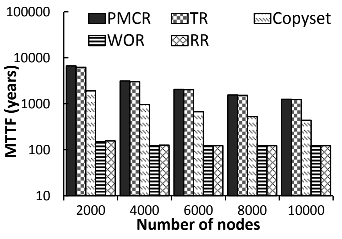
Fig. 15: Performance on storage cost of various methods.

We then use Formula (18) to calculate the bandwidth cost for PMCR. For RR, Copyset and TR, we use Formula (18) without considering compression. For WOR, we use Formula (18) with the consideration of compression. Figure 14(a) and Figure 14(b) show the relationship between the bandwidth cost and the number of data objects in the Facebook and HDFS environments, respectively. We observe that the bandwidth cost increases as the number of data objects increases. This is because more data objects lead to more data transfers for data updates and for data requests, which results in higher bandwidth cost. We also see that the bandwidth cost follows PMCR<WOR<TR≈Copyset≈RR. PMCR and WOR generate lower bandwidth cost than TR, Copyset and RR because they use compression and deduplication to reduce the data size in storage, which can reduce the bandwidth cost for data transfer.

We then use Formula (7) to calculate the storage cost for PMCR. For RR and Copyset, we use Formula (7) without considering compression or the selection of different storage mediums for storing data objects. For WOR, we use Formula (7) with the consideration of compression and without the selection of different storage mediums for storing data in the backup tier. For TR, we use Formula (7) without considering compression but with the selection of different storage mediums for storing replicas in the backup tier. Figure 15(a) and Figure 15(b) show the relationship between the storage cost and the number of data objects in the Facebook and HDFS environments, respectively. We see that the storage cost increases as the number of data objects increases because the more the data objects, the more storage resource needed for storing the data objects. We also see that the storage cost follows PMCR<WOR<TR<Copyset≈RR. TR has lower storage cost than Copyset and RR and higher storage cost than WOR and PMCR. This is because TR uses less expensive storage medium to store the third replicas of data objects to reduce the storage cost, which is not considered in Copyset and RR. WOR utilizes data compression and data deduplication to reduce storage cost. PMCR has the lowest storage cost because PMCR considers

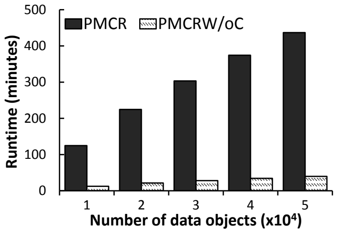


(a) Facebook

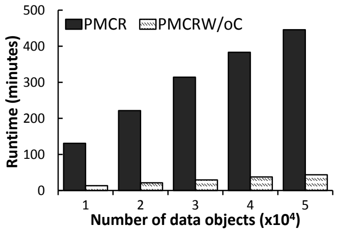


(b) HDFS

Fig. 16: Performance on MTTF of different methods with scatter width of 4.



(a) Facebook



(b) HDFS

Fig. 17: Performance on computational overhead on compression of PMCR.

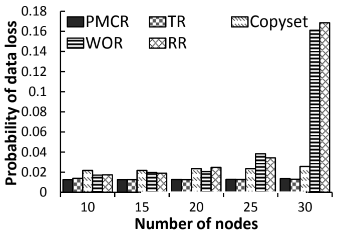
data popularity and uses compression to reduce the amount of data stored in the system, and also chooses less expensive storage mediums to store unpopular data objects.

Figure 16(a) and Figure 16(b) show the relationship between the MTTF (mean time to failure) and the number of nodes in the Facebook environment and HDFS environment, respectively. In Figure 16(a) and Figure 16(b), we see that the MTTF decreases as the number of nodes increases. This is because the probability of correlated machine failures increases as the number of nodes increases, which increases the probability that the nodes fail. We also see that the MTTF follows PMCR≈TR>Copyset≈RR≈WOR. The reason for the MTTF in PMCR, TR and Copyset being greater than RR and WOR is PMCR, TR and Copyset constrain the replicas of a data object to an FTS to reduce the probability of data loss caused by correlated machine failures and thus increase the availability of data object requests.

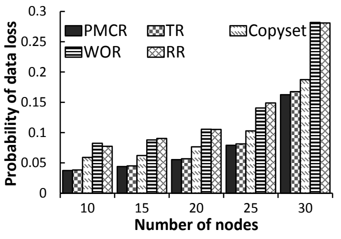
To test the computational overhead on compression of PMCR, we tested the runtime of PMCR and PMCR without compression (PMCRW/oC), a variant of PMCR in which compression is not used. Figure 17(a) and Figure 17(b) show the relationship between the computational overhead (runtime) and the number of objects in the Facebook environment and HDFS environment, respectively. In Figure 17(a) and Figure 17(b), we see that the runtime of PMCRW/oC is less than that of PMCR, and the runtime increases as the number of data objects increases. This is because the compression of data objects introduces additional time consumption and the larger the number of data objects the more the time required for compression. We also see that the runtime of PMCR increases faster than that of PMCRW/oC as the number of data objects increases.

5.2 Real-world Experimental Results

To further verify the performance of our method in the real-world environment, we conducted experiments on Amazon S3. We used three regions of Amazon S3 in the U.S. to generate geo-distributed storage datacenters. We created the same



(a) # of replicas R=3



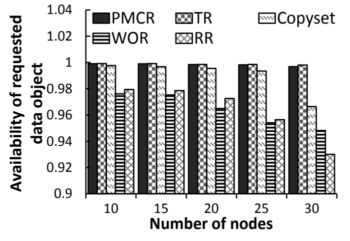
(b) # of replicas R=2

Fig. 18: Probability of data loss vs. the number of nodes on Amazon S3 with scatter width S=4.

number of buckets in each region and each bucket contains a data server. We varied the number of buckets from 10 to 30 with step size 5. We generated 50000 data objects. The sizes of data objects follow a normal distribution. We distributed the data objects to servers randomly. We used the distributions of read and writes from the FIU trace to generate reads and writes. The requests were generated from servers in Windows Azure eastern region. We consider the requests targeting each region with latency more than 100ms as failed requests due to unavailable data objects. In the test, N is the number of simulated data servers. We used the actual price of the data access of Amazon S3 [51] to calculate the storage cost and the bandwidth cost.

We first measure the probability of data loss for each method. Figure 18(a) and Figure 18(b) show the relationship between the probability of data loss and the number of nodes on Amazon S3 with $R = 3$ and $R = 2$, respectively. We see that the probability of data loss increases as the number of nodes increases. We also see that the probability of data loss approximately follows PMCR≈TR<Copyset<WOR≈RR. Both our numerical result and real-world experimental result confirm that PMCR and TR generate the lowest probability of data loss. PMCR and TR generate relatively lower probability of data loss than Copyset because they separate the primary data from the backup data by storing the backup data on a remote site, which further reduces the correlation in failures between nodes in the primary tier and the backup tier [21]. Copyset generates lower probability of data loss than WOR and RR. The reason is that Copyset constraints the replica nodes of every chunk to a single Copyset and reduces probability of data loss in correlated machine failures. WOR and RR cannot handle correlated machine failures and thus have higher probability of data loss. By examining Figure 18(a) and Figure 18(b), we find that the probability of data loss in Figure 18(b) is higher than that in Figure 18(a). This is because fewer replicas for a chunk lead to a higher probability that all the servers storing the chunk fail concurrently, hence a higher probability of data loss.

Figure 19(a) and Figure 19(b) show the relationship between the availability of requested data objects and the number of nodes on Amazon S3 with $R = 3$ and $R = 2$, respectively. We see that the availability of requested data objects decreases as the number of nodes increases. We also see that the availability of requested data objects follows PMCR≈TR>Copyset>RR≈WOR due to the same reasons explained in Figure 13. Comparing Figure 19(a) and Figure 19(b), we find that the availability of requested data objects in Figure 19(a) is higher than that in Figure 19(b). The reason is that


(a) # of replicas R=3

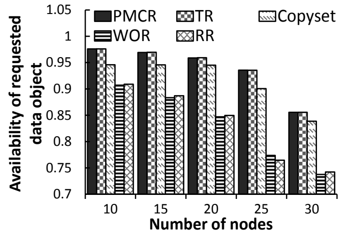
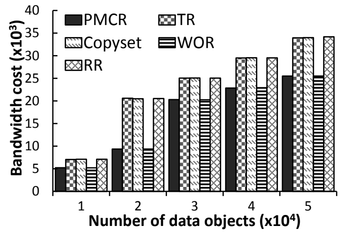

(b) # of replicas R=2

Fig. 19: Availability of requested data objects vs. the number of nodes on Amazon S3 with scatter width S=4.


(a) # of replicas R=3

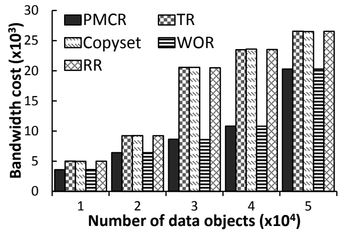

(b) # of replicas R=2

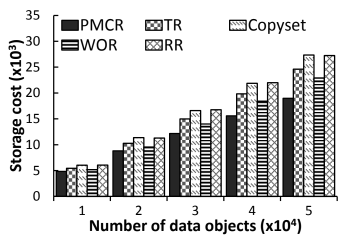
Fig. 20: Performance on bandwidth cost of various methods on Amazon S3.

the more the replicas for a chunk, the lower the probability that all the machines storing the chunk fail concurrently, leading to higher availability of requested data objects.

Figure 20(a) and Figure 20(b) show the relationship between the bandwidth cost and the number of data objects on Amazon S3 with R=3 and R=2, respectively. We observe that the bandwidth cost increases as the number of data objects increases due to the same reasons explained in Figure 14. We also see that the bandwidth cost follows PMCR≈WOR<TR≈Copyset≈RR. TR generates higher bandwidth cost than WOR and PMCR. This is because WOR and PMCR compress data objects and reduce the size of data for transfer for data requests and updates, and therefore reduce the bandwidth cost. Both the numerical result in Figure 14 and the real-world experimental result in Figure 20 indicate that compression (with deduplication) in replication is effective in reducing bandwidth cost. By examining Figure 20(a) and Figure 20(b), we see that the bandwidth cost increases as the number of replicas for each data object increases. This is because more replicas for each data object lead to more data transfers for data requests and updates.

Figure 21(a) and Figure 21(b) depict the relationship between the storage cost and the number of data objects on Amazon S3 with R=3 and R=2, respectively. We see that the storage cost increases as the number of data objects increases due to the same reasons explained in Figure 15. We also find that the storage cost follows PMCR<WOR<TR<Copyset≈RR. WOR generates higher storage cost than PMCR and lower storage cost than TR, Copyset and RR. This is because PMCR and WOR compress data objects, which reduces the storage cost, but other methods do not use compression. Moreover, PMCR considers data popularity neglected in all the other methods, and chooses less expensive storage media for storing unpopular data objects, which further reduces the storage cost. Comparing Figure 21(a) and Figure 21(b), we see that the storage cost increases as the number of replicas for each data object increases. The reason is that more replicas for each data object lead to higher storage consumption for storing data objects.

Figure 22(a) and Figure 22(b) shows the relationship between the MTTF and the number of nodes on Amazon S3 when $R=3$ and $R=2$, respectively. We find that the MTTF follows PMCR≈TR>Copyset>WOR≈RR. The results approximately conform the numerical results. Copyset, TR and PMCR have larger MTTF than WOR and RR because they constrain the replica nodes of every chunk in a single FTS, which reduces the failures in correlated machine failures. Copyset has relatively smaller MTTF than TR and PMCR because TR and PMCR separate the primary data from the the backup data and store the backup data in a remote site, which reduces the correlation in failures between nodes in the primary tier and the backup tier.


(a) # of replicas R=3

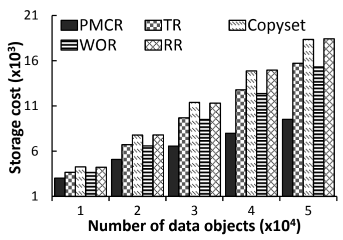
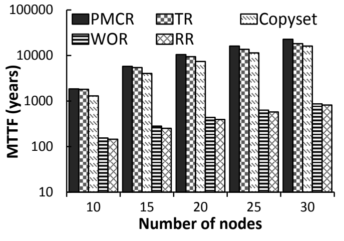

(b) # of replicas R=2

Fig. 21: Performance on storage cost of various methods on Amazon S3.


(a) # of replicas R=3

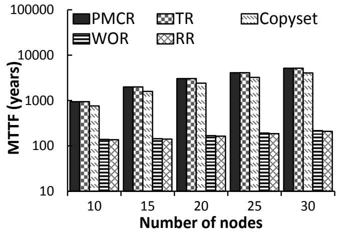

(b) # of replicas R=2

Fig. 22: Performance on MTTF of different methods with scatter width of 4 on Amazon S3.

6 Related Work

Many methods have been proposed to prevent data loss caused by correlated or non-correlated machine failures. Zhong *et al.* [9] assumed independent machine failures, and proposed a model that achieves high expected service availability under given space constraint, object request popularities, and object sizes. However, this model does not consider correlated machine failures and hence cannot handle such failures. Nath *et al.* [8] identified a set of design principles that system builders can use to tolerate failures. Cidon *et al.* [11] proposed Copyset Replication to reduce the frequency of data loss caused by correlated machine failures by limiting the replica nodes of many chunks to a single Copyset. Chun *et al.* [4] proposed the Carbonite replication algorithm for keeping data durable at a low cost. Carbonite ensures that creating new copies of data objects is faster than permanent disk failures. Cidon *et al.* [21] proposed a Tiered Replication that splits the cluster into a primary tier and abackup tier. The first two replicas of the data are stored on the primary tier, which is used for protect against independent node failures; the third replica is stored on the backup tier, which is used to protect against correlated failures. However, these methods do not try to reduce the storage cost and bandwidth cost caused by replication.

There is a large body of work on enhancing data availability and durability. Renesse *et al.* [52] proposed chain replication to coordinate clusters of fail-stop storage servers for supporting large-scale storage services that exhibit high throughput and availability without sacrificing strong consistency guarantees. Almeida *et al.* [53] proposed ChainReaction, a Geo-distributed key-value datastore, to offer causal+ consistency, with high performance, fault-tolerance, and scalability. Zhang *et al.* [54] proposed Mojim to provide the reliability and availability in large-scale storage systems while preserving the performance of non-volatile main memory. Mojim uses a two-tier architecture in which the primary tier contains a mirrored pair of nodes and the secondary tier contains one or more secondary backup nodes with weakly consistent copies of data. Kim *et al.* [55] proposed SHADOW systems to provide high availability. SHADOW systems push the task of managing database replication into the underlying storage service, and provide write offloading to free the active database system from the need to update the persistent database. Colgrove *et al.* [56] presented Purity, a all-flash enterprise storage system to support compression, deduplication and high-availability. Specifically, Purity leverages flash's ability to perform fast random reads and sequential writes by compressing data and storing a single instance of duplicate blocks written to different logical addresses. However, these works fail to consider data popularity to reduce the storage cost and bandwidth cost without compromising data request delay greatly.

In order to reduce the storage cost and bandwidth cost caused by replication, many methods have been proposed. Shilane *et al.* [49] proposed a new method for replicating backup datasets across a wire area network (WAN). The method can eliminate duplicate regions of files (deduplication) and also compress similar regions of files with Delta compression. The method leverages deduplication locality to also find similarity matches used for delta compression. Puttaswamy *et al.* [57] proposed FCFS, a storage solution that drastically reduces the cost of operating a file system in the cloud. FCFS integrates multiple storage services and dynamically adapts the storage volume sizes of each service to provide a cost-efficient solution with provable performance bounds. However, these methods do not consider data popularity to reduce the storage cost and bandwidth cost. Also, these methods neglect correlated machine failures, which can result in data loss in such failures.

To resolve the problems in the existing replication schemes, we propose PMCR that can effectively handle both correlated and non-correlated machine failures and also considers different file popularities to increase data durability and availability and reduce the bandwidth cost and storage cost without compromising data request delay greatly.

7 Conclusion

Previous replication schemes for cloud storage systems consider correlated machine failures or non-correlated machine failures to reduce data loss. However, although data replicas bring about additional storage and bandwidth costs, few methods aim to maximize data durability and availability while reducing the cost caused by replication (i.e., storage cost and bandwidth cost) with the consideration of data popularity. In this paper, in order to improve data durability and availability, and meanwhile reduce costs caused by replication, we propose a popularity-aware multi-failure resilient and cost-effective replication scheme (PMCR). PMCR classifies data into hot data, warm data and cold data based on the data popularity. PMCR puts the first two replicas of data objects to primary tier and puts the third replicas to backup tier. The replicas of the same data are put into one fault-tolerant set to handle correlated failures. PMCR uses SC for read-intensive data and uses DC for write-intensive data to compress the third replicas of warm data and cold data in the backup tier to reduce both storage cost and bandwidth cost. Our extensive numerical analysis and real-world experimental results on Amazon S3 show that PMCR outperforms other replication schemes in different performance metrics. In the future, we will further consider network failures to further reduce the data loss and improve the data durability. Also, we will consider the effects of node joining and node leaving. Further, we will consider energy consumption of machines and design a replication scheme to save energy.

Acknowledgment

This research was supported in part by U.S. NSF grants OAC-1724845, ACI-1719397 and CNS-1733596, and Microsoft Research Faculty Fellowship 8300751. An early version of this work was presented in the Proceedings of Big Data 2016 [58]. We would like to thank Dr. Rajkumar Buyya, Dr. Kuang-Ching Wang, Dr. James Z. Wang, Dr. Adam Hoover, Dr. Wingyan Chung, Dr. Svetlana Poznanović and Dr. Warren Adams for their help on this work.

References

[1] "Amazon S3," http://aws.amazon.com/s3 [accessed in Jan. 2016].
[2] Google cloud storage. http://cloud.google.com/storage [accessed in Jan. 2016].
[3] "Windows azure," http://www.microsoft.com/windowsazure [accessed in Jan. 2016].
[4] B. Chun, F. Dabek, A. Haeberlen, E. Sit, H. Weatherspoon, M. Kaashoek, J. Kubiatowicz, and R. Morris, "Efficient replica maintenance for distributed storage systems," in *Proc. of NSDI*, 2006.
[5] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's highly available key-value store," in *Proc. of SOSP*, 2007.
[6] R. Chansler, "Data availability and durability with the hadoop distributed file system," *The USENIX Magazine*, 2012.
[7] J. Dean, "Evolution and future directions of large-scale storage and computation systems at google," in *Proc. of ACM SoCC*, 2010.
[8] S. Nath, H. Yu, P. Gibbons, and S. Seshan, "Subtleties in tolerating correlated failures in wide-area storage systems," in *Proc. of NSDI*, San Jose, CA, May 2006, pp. 225–238.
[9] M. Zhong, K. Shen, and J. Seiferas, "Replication degree customization for high availability," in *Proc. of EuroSys*, 2008.
[10] A. Haeberlen, A. Mislove, and P. Druschel, "Glacier: Highly durable, decentralized storage despite massive correlated failures," in *Proc. of NSDI*, 2005.
[11] A. Cidon, S. Rumble, R. Stutsman, S. Katti, J. Ousterhout, and M. Rosenblum, "Copysets: Reducing the frequency of data loss in cloud storage," in *Proc. of USENIX ATC*, 2013.
[12] J. Liu and H. Shen, "A low-cost multi-failure resilient replication scheme for high data availability in cloud storage," in *Proc. of HiPC*, 2016, pp. 242–251.

[13] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in *Proc. of MSST*, 2010.

[14] D. Borthakur, J. Gray, J. Sarma, K. Muthukkaruppan, N. Spiegelberg, H. Kuang, K. Ranganathan, D. Molkov, A. Menon, S. Rash, R. Schmidt, and A. Aiyer, "Apache Hadoop goes realtime at Facebook," in *Proc. of SIGMOD*, 2011.

[15] S. Balakrishnan, R. Black, A. Donnelly, P. England, A. Glass, D. Harper, S. Legtchenko, A. Ogus, E. Peterson, and A. Rowstron, "Pelican: A building block for exascale cold data storage," in *Proc. of OSDI*, 2014, pp. 351–365.

[16] M. Ghosh, A. Raina, L. Xu, X. Qian, I. Gupta, and H. Gupta, "Popular is cheaper: Curtailing memory costs in interactive analytics engines," in *Proc. of EuroSys*, 2018.

[17] G. Ananthanarayanan, S. Agarwal, S. Kandula, A. Greenberg, I. Stoica, D. Harlan, and E. Harris, "Scarlett: Coping with skewed content popularity in mapreduce clusters," in *Proc. of EuroSys*, Salzburg, April 2011.

[18] F. André, A. Kermarrec, E. Merrer, N. Scouarnec, G. Straub, and A. Kempen, "Archiving cold data in warehouses with clustered network coding," in *Proc. of EuroSys*, 2014.

[19] "March, a. storage pod 4.0: Direct wire drives - faster, simpler, and less expensive," http://blog.backblaze.com/2014/03/19 /backblaze-storage-pod-4/, March 2014 [accessed in Jan. 2016].

[20] G. A. N. Yasa and P. C. Nagesh, "Space savings and design considerations in variable length deduplication," *SIGOPS Oper. Syst. Rev.*, vol. 46, no. 3, pp. 57–64, 2012.

[21] A. Cidon, R. Escriva, S. Katti, M. Rosenblum, and E. G. Sirer, "Tiered replication: A cost-effective alternative to full cluster geo-replication," in *Proc. of ATC*, 2015, pp. 31–43.

[22] D. Ongaro, S. Rumble, R. Stutsman, J. Ousterhout, and M. Rosenblum, "Fast crash recovery in ramcloud," in *Proc. of SOSP*, 2011, pp. 29–41.

[23] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *Proc. of SOSP*, 2003, pp. 29–43.

[24] B. Calder, J. Wang, A. Ogus, N. Nilakantan, A. Skjolsvold, S. McKelvie, Y. Xu, S. Srivastav, J. Wu, H. Simitci, J. Haridas, C. Uddaraju, H. Khatri, A. Edwards, V. Bedekar, S. Mainali, R. Abbasi, A. Agarwal, M. F. u. Haq, M. I. u. Haq, D. Bhardwaj, S. Dayanand, A. Adusumilli, M. McNett, S. Sankaran, and K. M. L. Rigas, "Windows azure storage: a highly available cloud storage service with strong consistency," in *Proc. of SOSP*, 2011.

[25] Y. Zhang, C. Guo, D. Li, R. Chu, H. Wu, and Y. Xiong, "Cubicring: Enabling one-hop failure detection and recovery for distributed in-memory storage systems," in *Proc. of NSDI*, 2015.

[26] H. Abu-Libdeh, R. Renesse, and Y. Vigfusson, "Leveraging sharding in the design of scalable replication protocols," in *Proc. of SoCC*, 2013.

[27] S. Y. Ko, I. Hoque, B. Cho, and I. Gupta, "Making cloud intermediate data fault-tolerant," in *Proc. of SoCC*, 2010.

[28] N. Bonvin, T. Papaioannou, and K. Aberer, "A self-organized, fault-tolerant and scalable replication scheme for cloud storage," in *Proc. of SoCC*, Indianapolis, 2010.

[29] D. Ford, F. Labelle, F. I. Popovici, M. Stokely, V.-A. Truong, L. Barroso, C. Grimes, and S. Quinlan, "Availability in globally distributed storage systems," in *Proc. of OSDI*, 2010.

[30] S. Welham, S. Gezan, S. Clark, and A. Mead, *Statistical Methods in Biology: Design and Analysis of Experiments and Regression*. CRC Press, 2014.

[31] "Cloud vps. https://www.cloudvps.nl/ [accessed in Jan. 2016]."

[32] R. Koller and R. Rangaswami, "I/o deduplication: Utilizing content similarity to improve i/o performance," in *Proc. of FAST*, 2010.

[33] N. Jain, M. Dahlin, and R. Tewari, "Taper: Tiered approach for eliminating redundancy in replica synchronization," in *Proc. of USENIX FAST*, 2005, pp. 281–294.

[34] W. Xia, H. Jiang, D. Feng, and Y. Hua, "Silo: A similarity-locality based near-exact deduplication scheme with low ram overhead and high throughput," in *Proc. of ATC*, 2011.

[35] G. You, S. Hwang, and N. Jain, "Scalable load balancing in cluster storage systems," in *Proc. of Middleware*, 2011.

[36] Y. Fu, H. Jiang, and N. Xiao, "A scalable inline cluster deduplication framework for big data protection," in *Proc. of Middleware*, 2012.

[37] N. Bonvin, T. Papaioannou, and A. Aberer, "Autonomic sla-driven provisioning for cloud applications," in *Proc. of CCGRID*, 2011.

[38] K. Chen and H. Shen, "Dsearching: Distributed searching of mobile nodes in dtns with floating mobility information," in *Proc. of INFOCOM*, 2014.

[39] X. Lin, G. Lu, F. Douglis, P. Shilane, and G. Wallace, "Migratory compression: Coarse-grained data reordering to improve compressibility," in *Proc. of USENIX FAST*, 2014.

[40] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–226, 1970.

[41] J. J. S. Houghten, L. Thiel and C. Lam., "There is no (46, 6, 1) block design*," *Journal of Combinatorial Designs*, vol. 9, no. 1, pp. 60–71, 2001.

[42] P. Kaski and P. Östergård, "There exists no (15, 5, 4) RBIBD," *Journal of Combinatorial Designs*, vol. 9, pp. 227–232, 2001.

[43] L. Xu, A. Pavlo, S. Sengupta, J. Li, and G. Ganger, "Reducing replication bandwidth for distributed document databases," in *Proc. of SoCC*, 2015.

[44] D. Arteaga and M. Zhao, "Client-side flash caching for cloud systems," in *Proc. of ACM SYSTOR*, Haifa, June 2014.

[45] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, "Oceanstore: An architecture for global-scale persistent storage," in *Proc. of ASPLOS*, 2000.

[46] M. Wittie, V. Pejovic, L. Deek, K. Almeroth, and B. Zhao, "Exploiting locality of interest in online social networks," in *Proc. of CoNEXT*, Philadelphia, 2010.

[47] S. Acedański, S. Deb, M. Médard, and R. Koetter, "How good is random linear coding based distributed networked storage," in *WINMEE, RAWNET and NETCOD*, 2005.

[48] Intelligent block placement policy to decrease probability of data loss. https://issues.apache.org/jira/browse/HDFS-1094 [accessed in Jan. 2016].

[49] P. Shilane, M. Huang, G. Wallace, and W. Hsu, "Wan optimized replication of backup datasets using stream-informed delta compression," in *Proc. of FAST*, 2012.

[50] V. Rawat, "Reducing failure probability of cloud storage services using multi-clouds," in *Proc. of CoRR*, 2013.

[51] "Amazon S3 Pricing," http://aws.amazon.com/s3/pricing/ [accessed in Jan. 2016].

[52] R. Renesse and F. Schneider, "Chain replication for supporting high throughput and availability," in *OSDI*, 2004.

[53] S. Almeida, J. Leitao, and L. Rodrigues, "Chainreaction: a causal+ consistent datastore based on chain replication," in *Proc. of EuroSys*, Prague, 2013, pp. 85–98.

[54] Y. Zhang, J. Yang, A. Memaripour, and S. Swanson, "Mojim: A reliable and highly-available non-volatile memory system," in *Proc. of ACM ASPLOS*, Istanbul, 2015.

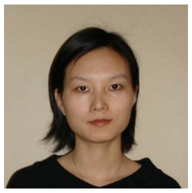
[55] J. Kim, K. Salem, K. Daudjee, A. Aboulnaga, and X. Pan, "Database high availability using shadow systems," in *Proc. of SoCC*, 2015.

[56] J. Colgrove, J. D. Davis, J. Hayes, E. L. Miller, C. Sandvig, R. Sears, A. Tamches, N. Vachharajani, and F. Wang, "Purity: Building fast, highly-available enterprise flash storage from commodity components," in *Proc. of SIGMOD*, 2015.

[57] K. P. N. Puttaswamy, T. Nandagopal, and M. Kodialam, "Frugal storage for cloud file systems," in *Proc. of EuroSys*, 2012.

[58] J. Liu and H. Shen, "A popularity-aware cost-effective replication scheme for high data durability in cloud storage," in *Proc. of IEEE BigData*, Washington D.C., 2016.

Jinwei Liu Jinwei Liu received the MS degree in Computer Science from Clemson University, SC, USA and University of Science and Technology of China, China. He received his Ph.D. degree in Computer Engineering from Clemson University, SC, USA, in 2016. He was a Research Associate at University of Virginia in 2017. He is currently a Postdoctoral Associate at University of Central Florida. His research interests include cloud computing, big data, machine learning and data mining, wireless sensor networks, social networks, HPC and IoT. He was the member of the Program Committees of several international conferences. He is a member of the IEEE and the ACM.



Haiying Shen received the BS degree in Computer Science and Engineering from Tongji University, China in 2000, and the MS and Ph.D. degrees in Computer Engineering from Wayne State University in 2004 and 2006, respectively. She is currently an Associate Professor in the Computer Science Department at the University of Virginia. Her research interests include cloud computing and cyber-physical systems. She was the Program Co-Chair for a number of international conferences and member of the Program Committees of many leading conferences. She is a Microsoft Faculty Fellow of 2010, a senior member of the IEEE and a member of the ACM.

Husnu S. Narman received his B.S. degree in Mathematics from Abant Izzet Baysal University, Turkey, in 2006, M.S. degree in Computer Science from University of Texas at San Antonio, San Antonio TX, USA in 2011, and PhD degree in Computer Science from University of Oklahoma, Norman OK, USA, in 2016. Currently, he is a faculty member at Marshall University, Huntington WV, USA. His research interests include queuing theory, network management, network topology, Internet of Things, LTE and Cloud Computing.

Zongfang Lin Zongfang Lin received the BS from Sun Yat-sen University, the ME in Information Science from Peking University, and the MS in Computer Science from University of Massachusetts, Amherst, and reached PhD/ABD in Computer Science with academic excellence at Amherst before leaving for his technical startup. He has worked at the United Nations, and Microsoft, and founded NConnex Inc. He is currently a principal architect at Huawei US R&D Center. His research interests include distributed and parallel computing, deep learning, and software engineering. He is a member of the IEEE and the ACM.

Zhuozhao Li Is currently a postdoc in Department of Computer Science at University of Chicago. He earned his Ph.D. in Department of Computer Science at University of Virginia, 2018. He received the B.S. degree in Optical Engineering from Zhejiang University, China in 2010, and the M.S. degree in Electrical Engineering from University of Southern California in 2012. His research interests include distributed computer systems, with an emphasis on cloud computing, resource management in cloud networks, and parallel computing.