A Popularity-aware Cost-effective Replication Scheme for High Data Durability in Cloud Storage

Jinwei Liu* and Haiying Shen[†]
*Department of Electrical and Computer Engineering, Clemson University, Clemson, SC 29634, USA
[†]Department of Computer Science, University of Virginia, Charlottesville, VA 22904, USA
jinweil@clemson.edu, hs6ms@virginia.edu

***Abstract*—Cloud storage system usually experiences data loss, hindering data durability. Three-way random replication is commonly used to prevent data loss in cloud storage systems. However, it cannot effectively handle correlated machine failures. 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 Similar Compression method for read-intensive data and uses the Delta Compression method for write-intensive data to reduce storage cost and bandwidth cost. As a result, these costs are reduced and data durability and availability are enhanced without compromising data request delay greatly. Extensive experiment results based on trace parameters show that PMCR achieves high data durability, low probability of data loss, and low storage cost and bandwidth cost compared to previous replication schemes.**

I. Introduction

Cloud providers, such as Amazon S3 [1], Google Cloud Storage (GCS) [2] and Windows Azure [3] offer storage as a service. It is important for cloud providers to reduce Service Level Agreement (SLA) violations to provide high quality of service and reduce the associated penalties [4–6]. High data durability is usually required 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) [7].

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) fail concurrently due to the common failure causes [8–10] (e.g., cluster power outages, Denial-of-Service attacks), and this type of failures often occur in large-scale storage systems [8, 11–13]. Significant data loss is caused by correlated machine failures [8], which has been documented by Yahoo! [14], LinkedIn [8] and Facebook [15]. Non-correlated machine failures refer to the events in which nodes fail individually (e.g., individual disk failure). Usually, non-correlated machine failures are caused by factors such as different hardware/software compositions and configurations [12, 16], and varying network access abilities.

To enhance data durability, data replication is commonly used in cloud storage systems. Due to highly skewed data popularity distributions [17], popular data with considerably higher request frequency (referred to as *hot data*) [18] could generate heavy load on some nodes, which may result in data unavailability at a time. Availability means that the requested data objects will be able to be returned to users [7]. Actually, much of the data stored in a cloud system is rarely read (commonly referred to as *cold data* [18, 19]). Replicas of cold data waste the storage resource and generate considerable storage cost and bandwidth cost (for data updates, data requests and failure recovery) [18, 20] that outweigh their effectiveness on enhancing data durability. Thus, it is important to compress and deduplicate unpopular data and store them in low-cost storage medium.

Random replication has been widely used in cloud storage systems [8, 9]. Cloud storage systems, such as Hadoop Distributed File System (HDFS) [14], Google File System (GFS) [21] and Windows Azure [22] use random replication to replicate their data in three servers randomly selected from different racks to prevent data loss in a single cluster [8, 9]. However, the three-way random replication cannot well handle correlated machine failures because data loss occurs if any combination of three nodes fail simultaneously [8]. To handle this problem, Copyset Replication [8] and Tiered Replication [9] have been proposed. However, both methods do not try to leverage data popularity to substantially reduce storage cost or bandwidth cost caused by replication.

To address the above issues, 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. We summarize the contributions of this work below.

- PMCR replicates the first two replicas of each data chunk in primary tier, and replicates the third replica in remote backup tier. The three replicas of each data chunk are stored in one Copyset, which can handle correlated failures [8]. As a result, PMCR can handle both correlated and independent failures.
- PMCR classifies data into hot data, warm data and cold data based on data popularity. It compresses the third replicas of warm data and cold data in the backup tier. For read-intensive data, PMCR uses the Similar Compression (SC), which lever-

ages the similarities among replica chunks and removes redundant replica chunks; for write-intensive data, PMCR uses the Delta Compression (DC), which records the differences of similar data objects and between sequential data updates. As a result, PMCR significantly reduces the storage cost and bandwidth cost caused by replication without compromising data durability and availability, as well as data request delay greatly.

- To further reduce the storage and bandwidth costs caused by replication, 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).
- We have conducted extensive trace-driven experiments to compare PMCR with other state-of-the-art replication schemes. The results show PMCR achieves high data durability, low data loss probability, storage and bandwidth cost.

The remainder of this paper is organized as follows. Section II presents the design for PMCR. Section III describes the analysis of system performance. Section IV presents the experiment results. Section V reviews the related work. Section VI concludes this paper with remarks on our future work.

II. System Design

Suppose there are m data objects and each data object is split into M partitions (i.e., chunks) in the cloud storage system [23]. A data object is lost if any of its partitions is lost [8]. Suppose there are N servers in the system. For analytical tractability, we assume that a server belongs to a rack, a room, a datacenter, 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 [23].

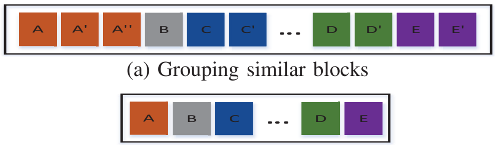
Problem Statement: Given data object request probabilities, data object sizes, and node failure probability, how to replicate the chunks of data objects so that the request failure probability, storage cost and bandwidth cost are minimized in both correlated failures and non-correlated failures?

A. PMCR Replication Scheme

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 (d_i) is measured by its visit frequency, i.e., the number of visits in a time epoch (denoted by v_i) [17, 18, 23–25]. That is, $\varphi_i(\cdot) = \alpha \cdot v_i$, where φ_i denotes d_i's popularity, 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 at epoch t:

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

where β $(0 < \beta < 1)$ is a coefficient. 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, with popularity rank between $(25\%, 50\%]$ as warm data, and with popularity rank between $(50\%, 100\%]$ as cold data.



(a) Grouping similar blocks

(b) Removing redundant copies

Fig. 1: Similar Compression.

PMCR sets thresholds for read rate and write rate, and it 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.

2) Replica Placement: PMCR first splits the nodes in the system into two tiers: primary tier and backup tier. For load balance, the number of nodes on the primary tier is twice of that on the backup tier [4, 26, 27]. To reduce the data loss caused by correlated machine failures, PMCR adopts the fault-tolerant set (FTS) [8] (i.e., Copyset). An FTS is a distinct set of servers that holds all replicas of a data object's chunk. We will explain the details of FTS in Section III-B1. PMCR then partitions the nodes and uses Balanced Incomplete Block Design (BIBD)-based method to generate FTSs.

In PMCR, 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. 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.

B. 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. Fig. 1 shows an example illustrating the process of grouping similar blocks and compressing the similar blocks together. In Fig. 1(a), similar blocks including (A, A', A"), (C, C'), (E, E') are grouped together. In Fig. 1(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 [28]. 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.

Also, PMCR extends the SC method originates from [28] 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. Fig. 2 shows an example of intra-file compression in a file. Similar blocks including (A, A'), (C, C'), (D, D'), (E, E') are marked in the same color. Fig. 3 shows an example of inter-file compression (deduplication). The blocks C and C' in the left

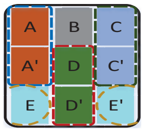


Fig. 2: Intra-file similarity.

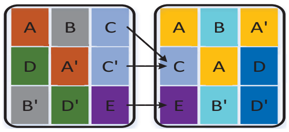


Fig. 3: Inter-file similarity.

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, 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.

C. Similarity Calculation

In this paper, we use the Bloom filter to detect the similarity between data blocks and extend this algorithm for detecting similarity between data chunks. 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.

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 that 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 [29]. 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.

III. Analysis of System Performance

A. Storage Cost Reduction

The replicas in the backup tier have lower read frequency compared to those 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 data popularity.

To reduce storage cost, we choose SSD to store the first two replicas of a hot data object 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.

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

$$I_c(d_i) = \begin{cases} 1, & \textit{if data object } d_i \textit{ is hot data} \\ 0, & \textit{if data object } d_i \textit{ is warm data or cold data} \end{cases} \tag{2}$$

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

$$s'_i = I_c(d_i) \cdot s_i/\gamma + (1 - I_c(d_i)) \cdot s_i \tag{3}$$

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

$$O_s = \sum_{i=1}^{m} (2 \cdot s_i + I_c(d_i) \cdot s_i + (1 - I_c(d_i)) \cdot s'_i) \tag{4}$$

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

Denote c_1, c_2, c_3 as the unit cost of SSD, disk and tape, respectively. The total storage cost (denoted by C_s) of PMCR is

$$C_s = \sum_{i=1}^{m} ((c_1 + (c_1 I_c(d_i) + c_2(1 - I_c(d_i))))s_i + c_3(I_c(d_i)s_i + (1 - I_c(d_i))s'_i)) \tag{5}$$

where s'_i is compressed data object d_i's storage consumption.

B. Data Durability Enhancement

1) Correlated Machine Failures: Recall that PMCR adopts FTS [8] to handle 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 FTSs 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 data loss probability caused by correlated machine failures can be minimized by minimizing the number of FTSs.

The probability of data loss in correlated machine failures (denoted by p_{cor}) is the ratio of the number of FTSs over the maximum possible number of sets. Based on [8], we have

$$p_{cor} = \#FTSs/max\{\#sets\} = \frac{S}{R-1}\frac{N}{R} / \binom{N}{R} \tag{6}$$

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 [8], the data loss probability caused by correlated machine failures in random replication can be obtained by substituting "$\#FTSs$" in Formula (6) by the number of FTSs created in random replication.

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 $(0 < p < 1)$. 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 chunk has three replicas. Hence, the expected probability of data loss due to uniform machine failure is

$$p_{uni} = (\sum_{j=1}^{m} M \cdot p^3)/m \tag{7}$$

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, 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 [12], 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, the expected probability of data loss caused by nonuniform machine failure is

$$p_{non} = (\sum_{j=1}^{m} M \cdot (\sum_{k=1}^{N} p_k/N)^3)/m \tag{8}$$

3) Correlated and Non-correlated Machine Failures: Denote F as the event that failure occurs, U_1, U_2 and U_3 as the event that correlated machine failures occur, the event that uniform machine failure occurs and the event that nonuniform machine failure occurs, respectively. Based on previous works [8, 12], 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 correlated and non-correlated machine failures is obtained as follows

$$P(F) = \sum_{i=1}^{3} P(F|U_i)P(U_i) \quad (\sum_{i=1}^{3} P(U_i) = 1) \tag{9}$$

where $P(F|U_1)$ (p_{cor} in Formula (6)), $P(F|U_2)$ (p_{uni} in Formula (7)) and $P(F|U_3)$ (p_{non} in Formula (8)) 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.

C. Bandwidth Cost Reduction

Based on [10, 23, 30, 31], the total bandwidth cost of all data objects caused by maintaining the consistency between the replicas of data objects [23, 32] can be calculated as follows:

$$C_b^c = \sum_{j=1}^{m} (3 \cdot M \cdot s^u E[\sum_{i,j} dis(S_i, S_j) \cdot \sigma]) \tag{10}$$

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 (called *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. We use the method in [10, 23] to compute the geographic distance between servers. σ is the average communication cost of a unit data per unit distance.

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 for recovering data, denoted by C_b^r, is

$$\begin{aligned} C_b^r =& (\sum_{i=1}^{m} (2 \cdot s_i)/\lfloor 2N/3 \rfloor \lceil 2NP(F)/3 \rceil + \sum_{i=1}^{m} (I_c(d_i)s_i \\ &+ (1 - I_c(d_i))s_i')/\lfloor N/3 \rfloor \lfloor NP(F)/3 \rfloor) \cdot \delta_d \end{aligned} \tag{11}$$

TABLE I: Parameters from publicly available data [8].

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

TABLE II: Parameter settings.

Parameter	Meaning	Setting
N	# of servers	1000-10000
M	# of chunks of a data object	50 [33]
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 [34]
m	# of data objects	10000-50000

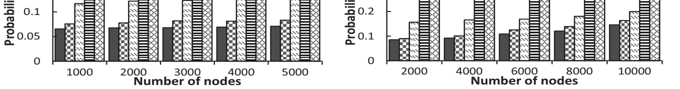
where s_i' and s_i are the size of data object d_i with and without compression, respectively, $\sum_{i=1}^{m}(2s_i)/\lfloor 2N/3 \rfloor$ and $\sum_{i=1}^{m}(I_c(d_i)s_i + (1-I_c(d_i))s_i')/\lfloor N/3 \rfloor$ are the average amount of data stored on a server in primary tier and backup tier for three-way replication, respectively. $\lceil 2NP(F)/3 \rceil$ and $\lfloor NP(F)/3 \rfloor$ are the number of failure nodes in primary tier and 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 (10) and (11), the total bandwidth cost caused by consistency maintenance and data recovery is

$$C_b = C_b^c + C_b^r \tag{12}$$

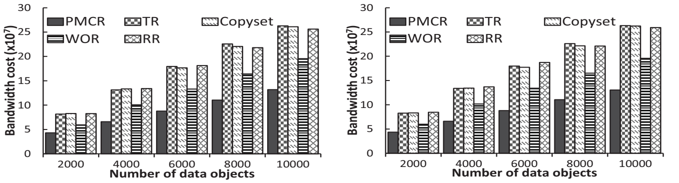
IV. Performance Evaluation

We conducted the experiments based on the parameters in [8] (TableI) under various scenarios.We compare our method with the other replication schemes: Random Replication (RR), Copyset Replication (Copyset) [8], Tiered Replication (TR) [9] and WAN Optimized Replication (WOR) [35]. 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,...). 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 chunk 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



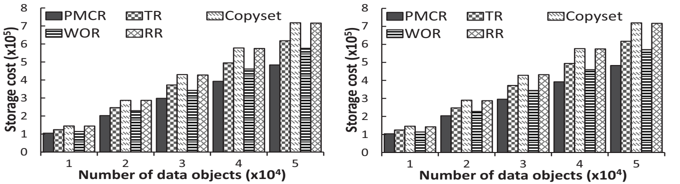
(a) Facebook (b) HDFS

Fig. 4: Probability of data loss vs. # of nodes.



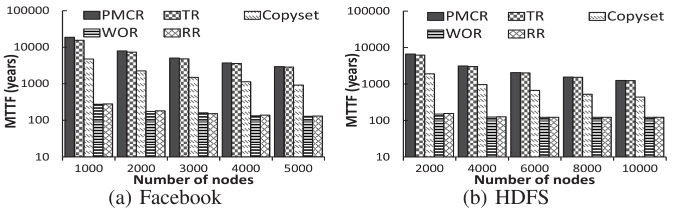
(a) Facebook (b) HDFS

Fig. 5: Bandwidth cost vs. # of data objects.



(a) Facebook (b) HDFS

Fig. 6: Storage cost of various methods.



(a) Facebook (b) HDFS

Fig. 7: MTTF of various methods.

set to 1% of the nodes in the system [8]. 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 [36]. Table II shows the parameter settings in the experiment unless otherwise specified.

We first calculate the probability of data loss for each method. We use Formula (9) to calculate the probability of data loss for PMCR and Formula (6) for Copyset. We use the method in [8] to calculate the data loss probability of random replication for RR and WOR, and use the method in [9] to calculate the data loss probability for TR. Fig. 4(a) and 4(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 chunk 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 [9]. 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 use Formula (12) to calculate the bandwidth cost for PMCR. For RR, Copyset and TR, we use Formula (12) without considering compression. For WOR, we use Formula (12) with the consideration of compression. Fig. 5(a) and 5(b) show the relationship between the bandwidth cost and the number of data objects. We 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 use Formula (5) to calculate storage cost for PMCR. For RR and Copyset, we use Formula (5) without considering compression or the selection of different storage mediums for storing data objects. For WOR, we use Formula (5) with considering compression and without the selection of different storage mediums for storing data in backup tier. For TR, we use Formula (5) without considering compression but with the selection of different storage mediums for storing replicas in backup tier. Fig. 6(a)-6(b) show the relationship between storage cost and the number of data objects. We see the storage cost follows PMCR<WOR<TR<Copyset≈RR because TR uses less expensive storage medium to store the third replicas of data objects to reduce 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 uses compression and deduplication to reduce the amount of data stored in the system, and 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.

Fig. 7 shows the results of MTTF (mean time to failure). In Fig. 7, we see that the MTTF decreases as the number of nodes increases 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. Copyset, TR and PMCR have larger MTTF than WOR and RR, and Copyset has relatively smaller MTTF than TR and PMCR due to the same reasons in Fig. 4.

V. Related Work

Many methods have been proposed to prevent data loss caused by correlated or non-correlated machine failures. Zhong *et al.* [12] assumed independent machine failures, and proposed a model that achieves high expected service availability. However, this model does not consider correlated machine failures and hence cannot handle such failures. Cidon *et al.* [8] proposed Copyset Replication to reduce the frequency of data loss caused by correlated machine failures by limiting the replica nodes of a chunk to a single Copyset. Chun *et al.* [7] proposed the Carbonite replication algorithm for keeping data durable at a low cost. Cidon *et al.* [9] proposed Tiered Replication that splits the cluster into a primary tier and a backup tier. The first two replicas of the data are stored on the primary tier to protect against independent failures; the third replica is stored on the backup tier to protect against correlated failures. However, these methods do not try to reduce storage cost and bandwidth cost caused by replication though data replicas bring about considerably high storage and bandwidth costs.

There is a large body of work on enhancing data availability and durability. Zhang *et al.* [37] proposed Mojim to provide the reliability and availability in large-scale storage systems. Colgrove *et al.* [4] presented Purity, an all-flash enterprise storage system to support compression, deduplication and high-availability. 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 storage cost and bandwidth cost caused by replication, many methods have been proposed. Shilane *et al.* [35] proposed stream-informed delta compression for replication of backup datasets across a wire area network (WAN). Puttaswamy *et al.* [38] proposed FCFS to reduce the cost of operating a file system in the cloud. 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 data popularities to increase data durability and availability and reduce the bandwidth cost and storage cost without compromising data request delay greatly.

VI. Conclusion

In this paper, in order to improve data durability and availability, and reduce costs caused by replication, we propose PMCR. PMCR classifies data into hot data, warm data and cold data based on 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 chunk are put into one FTS to handle correlated machine failures. PMCR uses SC for read-intensive data and DC for write-intensive data to compress the third replicas of warm data and cold data to reduce both storage cost and bandwidth cost. Our extensive experiment results show that PMCR outperforms other replication schemes in different performance metrics. In the future, we will further consider network failures to further reduce data loss and improve data durability. Also, we will consider the effects of node joining and node leaving. Further, we will consider energy consumption [39] of machines and design a replication scheme to save energy.

Acknowledgements

This research was supported in part by U.S. NSF grants NSF-1404981, IIS-1354123, CNS-1254006, IBM Faculty Award 5501145 and Microsoft Research Faculty Fellowship 8300751.

References

[1] Amazon S3. http://aws.amazon.com/s3 [accessed in Aug. 2016].

[2] Google cloud storage. http://cloud.google.com/storage.

[3] Windows azure. http://www.microsoft.com/windowsazure.

[4] 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.

[5] J. Liu, H. Shen, and H. Narman. CCRP: Customized cooperative resource provisioning for high resource utilization in clouds. In *Proc. of IEEE Big Data*, Washington D.C., 2016.

[6] J. Liu, H. Shen, and L. Chen. CORP: Cooperative opportunistic resource provisioning for short-lived jobs in cloud systems. In *Proc. of IEEE Cluster*, 2016.

[7] 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.

[8] 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*, pages 37–48, 2013.

[9] A. Cidon, R. Escriva, S. Katti, M. Rosenblum, and E. Gün Sirer. Tiered replication: A cost-effective alternative to full cluster geo-replication. In *Proc. of USENIX ATC*, pages 31–43, 2015.

[10] 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.

[11] S. Nath, H. Yu, P.B. Gibbons, and S. Seshan. Subtleties in tolerating correlated failures in wide-area storage systems. In *NSDI*, 2006.

[12] M. Zhong, K. Shen, and J. Seiferas. Replication degree customization for high availability. In *Proc. of EuroSys*, 2008.

[13] H. Shen, J. Liu, K. Chen, J. Liu, and S. Moyer. SCPS: A social-aware distributed cyber-physical human-centric search engine. *IEEE Transactions on Computers (TC)*, 64:518–532, 2015.

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

[15] 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 *SIGMOD*, 2011.

[16] J. Liu and H. Shen. Dependency-aware and resource-efficient scheduling for heterogeneous jobs in clouds. In *Proc. of CloudCom*, 2016.

[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, 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] 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.

[20] J. Liu, H. Shen, and X. Zhang. A survey of mobile crowdsensing techniques: A critical component for the internet of things. In *Proc. of ICCCN workshop on ContextQoS*, 2016.

[21] S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. In *Proc. of SOSP*, pages 29–43, 2003.

[22] 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. Haq, M. Haq, D. Bhardwaj, S. Dayanand, A. Adusumilli, M. McNett, S. Sankaran, K. Manivannan, and L. Rigas. Windows azure storage: a highly available cloud storage service with strong consistency. In *Proc. of SOSP*, 2011.

[23] N. Bonvin, T. Papaioannou, and K. Aberer. A self-organized, fault-tolerant and scalable replication scheme for cloud storage. In *SoCC'10*.

[24] L. Yan, K. Chen, H. Shen, and G. Liu. Mobilecopy: Resisting correlated node failures to enhance data availability in dtns. In *SECON*, 2015.

[25] B. Wu, H. Shen, and K. Chen. Exploiting active sub-areas for multi-copy routing in vdtns. In *Proc. of ICCCN*, 2015.

[26] A. Sarker, C. Qiu, H. Shen, A. Gil, J. Taiber, M. Chowdhury, J. Martin, M. Devine, and A. Rindos. An efficient wireless power transfer system to balance the state of charge of electric vehicles. In *ICPP*, 2016.

[27] B. Wu, H. Shen, and K. Chen. Dial: A distributed adaptive-learning routing method in vdtns. In *Proc. of IoTDI*, 2016.

[28] 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.

[29] N. Jain, M. Dahlin, and R. Tewari. Taper: Tiered approach for eliminating redundancy in replica synchronization. In *FAST*, 2005.

[30] 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.

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

[32] J. Liu, H. Shen, and H. Hu. Load-aware and congestion-free state management in network function virtualization. In *Proc. of ICNC*, 2017.

[33] S. Acedański, S. Deb, M. Médard, and R. Koetter. How good is random linear coding based distributed networked storage. In *NetCod*, 2005.

[34] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with cfs. In *Proc. of SOSP*, 2001.

[35] 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.

[36] Webserver workload. http://visa.lab.asu.edu/web/resources/traces/.

[37] Y. Zhang, J. Yang, A. Memaripour, and S. Swanson. Mojim: A reliable and highly-available non-volatile memory system. In *ASPLOS*, 2015.

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

[39] J. Liu, L. Yu, H. Shen, Y. He, and J. Hallstrom. Characterizing data deliverability of greedy routing in wireless sensor networks. In *Proc. of SECON*, Seattle, June 2015.