A Low-Cost Multi-Failure Resilient Replication Scheme for High Data Availability 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*—Replication is a common approach to enhance data availability in cloud storage systems. Previously proposed replication schemes cannot effectively handle both correlated and non-correlated machine failures while increasing the data availability with the limited resource. The schemes for correlated machine failures must create a constant number of replicas for each data object, which neglects diverse data popularities and cannot utilize the resource to maximize the expected data availability. Also, the previous schemes neglect the consistency maintenance cost and the storage cost caused by replication. It is critical for cloud providers to maximize data availability (hence minimize SLA violations) while minimizing cost caused by replication in order to maximize the revenue. In this paper, we build a nonlinear integer programming model to maximize data availability in both types of failures and minimize the cost caused by replication. Based on the model's solution for the replication degree of each data object, we propose a low-cost multi-failure resilient replication scheme (MRR). MRR can effectively handle both correlated and non-correlated machine failures, considers data popularities to enhance data availability, and also tries to minimize consistency maintenance cost and storage cost. Extensive numerical results from trace parameters and experiments from real-world Amazon S3 show that MRR achieves high data availability, low data loss probability and low consistency maintenance cost and storage cost compared to previous replication schemes.**

I. Introduction

Datacenter storage system (e.g., Hadoop Distributed File System (HDFS) [1], RAMCloud [2], Google File System (GFS) [3] and Windows Azure [4]) is an important component of cloud datacenters, especially for data-intensive services in this big data era. It is critical for cloud providers to reduce the violations of Service Level Agreements (SLAs) for tenants to provide high quality-of-service and avoid the associated penalties. For example, a typical SLA required from services that use Amazon's Dynamo storage system is that 99.9% of the read and write requests execute within 300ms [5]. Therefore, a storage system must guarantee data availability for different applications to comply to SLAs, which is, however, a non-trivial task. In many cloud services such as Amazon, Google App Engine and Windows Azure, the server failure probability is in the range of [1%, 10%] and the corresponding data availability is in the range of [99.9%, 99.99%] [6].

Data availability is usually influenced by data loss, which is typically caused by machine failures including correlated and non-correlated machine failures coexisting in storage systems. The former means multiple nodes fail (nearly) simultaneously, while the latter means nodes fail individually. Correlated machine failures often occur in large-scale storage systems [7]–[9] due to common failure causes (e.g., cluster power outages, workload-triggered software bug manifestations, Denial-of-Service attacks). For example, in cluster power outages [10], [11], a non-negligible percentage (0.5%-1%) of nodes [1], [10] do not come back to life after power is restored [12]. Correlated machine failures cause significant data loss [12], which have been documented by Yahoo! [1], LinkedIn [10] and Facebook [13]. Non-correlated machine failures [8] are caused by reasons such as different hardware/software compositions and configurations, and varying network access ability.

Replication is a common approach to reduce data loss and enhance data availability. The data popularity consideration in replication is critical to maximize the expected data availability[1]. Due to highly skewed data popularity distributions [16], popular data with considerably higher request frequency generates heavier load on the nodes, which may lead to data unavailability at a time. On the other hand, unpopular data with few requests wastes the resources for storing and maintaining replicas. Thus, an effective replication scheme must consider the diverse data popularities to use the limited resources (e.g., memory) [17], [18] to increase expected data availability.

Though creating more replicas for data objects improves data availability, it comes with higher consistency maintenance cost [19]–[21] and storage cost [22], [23]. In consistency maintenance, when data is updated, an update is sent to its replica nodes. In addition to the number of replicas, the consistency maintenance cost and storage cost are also affected by the geographic distance and storage media (e.g., disk, SSD, EBS), respectively. To reduce the cost, we need to minimize the number of replicas, limit the geographic distance of replica nodes and replicate the additional replicas beyond the required storage of applications to less expensive storage media while still provide SLA guarantee. Therefore, effective replication methods must maximize expected data availability (by considering both correlated and non-correlated machine failures and data popularity) and minimize the cost caused by replication (i.e., consistency maintenance cost and storage cost) [20], [24].

Random replication has been widely used in datacenter storage systems including HDFS, RAMCloud, GFS and Windows Azure. These systems partition each data object into chunks (i.e., partitions), each of which is replicated to a constant number of randomly selected nodes on different racks. Though ran-

[1]We use a service-centric availability metric: the proportion of all successful service requests over all requests [8], [14], [15].

dom replication can handle non-correlated machine failures, it cannot handle correlated machine failures well. The reason is that any combination of a certain number of nodes would form a set of nodes for storing all replicas of a data chunk and data loss occurs if all nodes in the set experience failures simultaneously [12]. Previously proposed data replication methods cannot effectively handle both correlated and non-correlated machine failures and utilize the limited resource to increase data availability simultaneously [7], [8], [12], [25]. Although many methods have been proposed to improve data availability [7], [8], [19], [25]–[27], they do not concurrently consider data popularities and the cost caused by replication to increase expected data availability and decrease cost.

As shown in Figure 1, a key problem here is how to achieve an optimal tradeoff between increasing data availability and reducing cost caused by replication with the ultimate goal of SLA compliance and revenue maximization for cloud service providers. To address the problem, we propose a low-cost multi-failure resilient replication scheme (MRR), which targets the application of distributed data-intensive services and storage system. MRR is more advantageous than previous replication schemes (e.g., Random Replication, Copyset Replication [12]) in that it can handle both correlated and non-correlated machine failures, and also jointly considers data popularity and the cost caused by replication. We summarize the contribution of this work below.

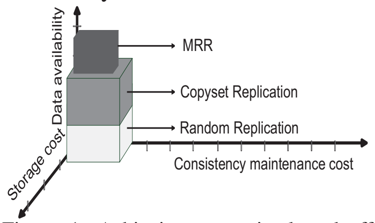


Figure 1: Achieving an optimal tradeoff among data availability, storage cost and consistency maintenance cost.

- We build a nonlinear integer programming (NLIP) model that aims to maximize expected data availability (in both types of failures) with the consideration of popularities and reduce the cost caused by replication. We then use Lagrange multipliers to derive a solution: the replication degree (i.e., the number of replicas) of each data object.
- Based on the solution, we propose MRR to handle both correlated and non-correlated machine failures. MRR partitions all nodes to different groups with each group responsible for replicating all data objects with the same replication degree. Then, MRR partitions nodes in a group into different sets [12]; each set consists of nodes from different datacenters within a certain geographic distance. The replicas of a chunk are stored in the nodes in one set, so data loss occurs only if these nodes experience failures simultaneously. Each data chunk is replicated to the corresponding storage medium based on its priority. MRR reduces the frequency of data loss by reducing the number of sets in a group, i.e., the probability that all nodes in a set fail.
- We have conducted extensive numerical analysis based on trace parameters and experiments on Amazon S3 [28] to compare MRR with other state-of-the-art replication schemes. Results show MRR achieves high data availability, low data loss probability and low storage and consistency maintenance cost.

The remainder of this paper is organized as follows. Section II reviews the related work. Section III presents our NLIP model. Section IV presents the design for MRR. Section V presents the numerical and experimental results. Section VI concludes this paper with remarks on our future work.

II. Related Work

Many methods have been proposed to handle non-correlated or correlated machine failures. Zhong *et al.* [8] assumed independent machines failures, and proposed a model that achieves high expected service availability. However, this model does not consider correlated machine failures hence cannot handle such failures. Nath *et al.* [7] identified a set of design principles that system builders can use to tolerate correlated failures. Cidon *et al.* proposed Copyset Replication [12] and Tiered Replication [29] to reduce the frequency of data loss caused by correlated machine failures by limiting the replica nodes of many chunks to a single copyset. These methods for correlated machine failures do not consider data popularity to minimize data loss probability in correlated and non-correlated machine failures. Unlike Copyset Replication and Tiered Replication, our proposed MRR reduces probability of data loss caused by both correlated and non-correlated machine failures with considering data popularity, and it derives diverse replication degrees for data objects with different popularities and thus increases the overall data object availability by creating more replicas for popular data objects and less for unpopular data objects. Moreover, MRR replicates data objects with considering reducing consistence maintenance cost and storage cost, which is critical for cloud providers to maximize the revenue. Thomson *et al.* [25] presented CalvinFS, a scalable distributed file system, which horizontally partitions and replicates file system metadata across a shared-nothing cluster of servers. However, CalvinFS cannot handle both correlated and non-correlated machine failures while minimizing cost (e.g., consistency maintenance cost and storage cost) caused by replication. The above previously proposed methods cannot handle both correlated machine failures and non-correlated machine failures while utilizing the limited resource to increase data availability. They neglect data popularity existing in current cloud storage system and thus cannot fully utilize the resource to increase data availability. Also, they do not concurrently consider minimizing the consistency maintenance cost and storage cost caused by replication.

There is a large body of work on enhancing data availability for distributed storage systems. Abu-Libdeh *et al.* [26] presented a replication protocol for datacenter services to provide strong consistency and high availability. Bonvin *et al.* [19] proposed a self-managed key-value store that dynamically allocates the resources of a data cloud to several applications in a cost-efficient and fair manner. Zhang *et al.* [30] presented Mojim to provide the reliability and availability in large-scale storage systems. Mojim uses a two-tier architecture in which the primary tier contains a mirrored pair of nodes and the secondary tier contains the secondary backup nodes with weakly consistent copies of data. Yu *et al.* [31] used a fixed number of replicas for every data object, and showed that the assignment of object replicas to machines plays a dramatic role in the availability of multi-object operations. However, most of these previous works neglect data object popularities when determining the number of replicas for each object, thus cannot satisfy the demands of popular data objects or fully utilize the limited resource. Although the works in [8], [16] consider object popularities, they do not give a solution for handling correlated machine failures, which is a key issue in achieving high availability in today's cloud datacenters [7]. In addition, they do not consider different storage medium prices (for reducing storage cost) or geographic distance in selecting nodes to store replicas, both of which affect the total cost of the storage systems.

Motivated by the problems in the existing works, we propose MRR that can effectively handle both correlated and non-correlated machine failures and also considers the factors indicated in the introduction section to maximize data availability and reduce the consistency maintenance cost and storage cost.

III. Nonlinear Integer Programming Model for MRR

A cloud storage system usually serves multiple applications simultaneously. Without loss of generality, we assume there are n applications in the cloud storage, and each application has m data objects [19]. Each data object belongs to only one application, and is split into M partitions [19] and the data object is lost if any of its partitions is lost [12]. Replication degree of a data object represents the number of replicas of the data object. We use D_{ij} to denote the jth data object belonging to application i (denoted by a_i). Let d_{ij} be the replication degree (number of replicas) of D_{ij}. The replicas of a partition of a data object are placed in a set of d_{ij} different nodes. Suppose there are N servers in the cloud. For analytical tractability, we assume a physical node (i.e., a server) belongs to a rack, a room, a datacenter, a country and a continent. To easily identify the geographic location of a physical node, each physical node has a label in the form of "continent-country-datacenter-room-rack-server" [19].

Problem Statement: Given data object request probabilities, data object sizes, space constraints for different applications, and thresholds for request failure probability, consistency maintenance cost and storage cost, what is the optimal replication degree for each data object, so that the request failure probability, consistency maintenance cost and storage cost are minimized in both correlated and non-correlated machine failures? Then, how to assign the chunk replicas of data objects to the nodes to achieve the objectives?

A. *Data Availability Maximization*

We maximize data availability by considering both machine failure probability and data object request probability (i.e., popularity). We minimize the data loss probability in both correlated and non-correlated machine failures. Different data objects may have different popularities, and then have different demands on the number of replicas to ensure data availability.

1) Correlated Machine Failures: To reduce data loss in correlated machine failures, we adopt the fault-tolerant set (FTS) concept from [12], which is a distinct set of servers holding all copies of a given partition. Each FTS is a single unit of failure. That is, when an FTS fails, at least one data object is lost. For correlated machine failures, the probability of data loss increases as the number of FTSs increases because the probability that the failed servers constitute at least one FTS increases (It is more likely that the failed servers include at least one FTSs). So we minimize the probability of data loss by minimizing the number of FTSs.

To this end, we can statically assign each server to a single FTS, and constrain the replicas of a partition to a randomly selected preassigned FTS. That is, we first place the primary replica (i.e., original copy) on a randomly selected server, and then place the secondary replicas on all the nodes in this server's FTS. However, this will lead to load imbalance problem in which some servers become overloaded while some servers are underloaded due to data storage. On the other hand, random replication that randomly chooses a replica holder has a higher probability of data loss because almost every new replicated partition creates a distinct FTS. To achieve a tradeoff, we use the approach in Copyset Replication [12], which assigns a server to a limited number of FTSs rather than a single FTS. Due to the overlap of FTSs, after the primary replica is stored in a randomly selected server, the FTS options that can store the secondary replicas are those that hold the server. The servers in these sets are options that can be used to store the secondary replicas. The number of these servers is called *scatter width* (denoted by S). For example, if the FTSs that hold server 1 are $\{1,2,3\}$, $\{1,4,5\}$, then when the primary replica is stored at server 1, the secondary replica can be randomly placed either on servers 2 and 3 or 4 and 5. In this case the scatter width equals to 4.

The probability of correlated machine failures equals the ratio of the number of FTSs over the maximum number of sets:

$$\#FTSs/max\{\#sets\} \tag{1}$$

To reduce the probability of data loss, Copyset Replication makes the replication scheme satisfy two conditions below:

- Condition 1: The FTSs overlap with each other by *at most* one server.
- Condition 2: The FTSs cover all the servers equally.

Copyset Replication uses the Balanced Incomplete Block Design (BIBD)-based method for any scatter width to create FTSs that satisfy both condition 1 and 2 and minimize #FTSs.[2]

We define a pair (X,A), where X is a set of servers in the system (i.e., $X=\{1,2,...,N\}$), and A is a collection of all FTSs in the system. Let N,R and λ be positive integers such that $N > R \geq 2$, BIBD for (N,R,λ) satisfies the following properties:

- Each FTS contains exactly R servers.
- Each pair of servers is contained in exactly λ FTSs.

When $\lambda=1$, the BIBD provides an optimal design for minimizing the number of FTSs for scatter width $S=N-1$. In this case, condition 1 ensures that each FTS increases the scatter width for its servers by exactly $R-1$ compared to the case when $\lambda=0$. Copyset Replication creates $\frac{S}{R-1}\frac{N}{R}$ FTSs. Then, the failure probability in correlated machine failures equals:

[2]In implementation, Copyset Replication uses random permutation to create FTSs.

$$p_{cor} = \frac{S}{R-1}\frac{N}{R}/\binom{N}{R} \quad (2)$$

In random replication, the number of FTSs created is [12]:

$$\begin{cases} \#FTSs = N\binom{S}{R-1}, & S < \frac{N}{2} \\ \#FTSs \approx \binom{N}{R}, & S \approx N \end{cases} \quad (3)$$

Based on Equs. (1), (3), we can calculate the probability of correlated machine failures in random replication.

We give an example to illustrate the process of generating FTSs. Consider a storage system with $N = 12$ servers, the size of FTS $R = 3$, and the scatter width $S = 4$. Using the BIBD-based method, the following solution of FTSs is created to achieve less number of FTSs.

$B_1 = \{0,1,2\}, B_2 = \{3,4,10\}, B_3 = \{6,7,8\}, B_4 = \{9,10,11\}$,
$B_5 = \{0,3,8\}, B_6 = \{1,4,7\}, B_7 = \{2,5,11\}, B_8 = \{5,6,9\}$.

For random replication, the #FTSs is 72. Hence, the probability of data loss caused by correlated machine failures is:

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

However, for BIBD-based method from Copyset Replication, the number of FTSs is 8, and the probability of data loss caused by correlated machine failures is much smaller:

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

There are many methods that can be used for constructing BIBDs, but no single method can create optimal BIBDs for any combination of N and R [32], [33]. Copyset Replication combines BIBD and random permutations to generate a non-optimal design that can accommodate any scatter width. By relaxing the constraint for the number of overlapping nodes from an exact number to at most the exact number, the probability of successfully generating FTSs increases. Since the scatter width should be much smaller than the number of nodes, MRR is likely to generate FTSs with at most one overlapping node [12]. MRR tries to minimize the replication degrees of data objects in order to limit the consistency maintenance cost and storage cost, the replication degrees should be not large or vary greatly. Smaller replication degrees generate smaller R values. Although sometimes it is not possible to generate the optimal BIBDs, BIBDs can be used as a useful benchmark to measure how close MRR is to the optimal scheme for specific values of scatter width [12].

2) Non-correlated Machine Failures: In non-correlated machine failures, the failures of machines are statistically independent of each other, and they can be categorized into uniform and nonuniform machine failures. In the scenario of uniform machine failures, each machine has the same probability to fail, denoted by p $(0 < p < 1)$. The data object is lost if any of its chunk (partition) is lost, and a chunk is lost only if all replicas of the chunk are lost. Hence, the expected probability of data loss in the uniform machine failure is:

$$p_{uni} = (\sum_{i=1}^{n}\sum_{j=1}^{m} M \cdot p^{d_{ij}})/(m \cdot n) \quad (4)$$

where M is the number of partitions (chunks) for each data object, n is the number of applications, and m is the number of data objects in each application.

In the scenario of nonuniform machine failures, each machine fails with different probabilities. Assume replicas of each data object are placed on machines with no concern for individual machine failures. Let $p_1, ..., p_N$ be the failure probabilities of N servers in the cloud, respectively. Based on [8], the expected data object failure probability is the same as that on uniform failure machines with per-machine failure probability equaling $\sum_{i=1}^{N} p_i/N$. We use p_{non} to denote the expected probability of data loss in nonuniform machine failures. Then,

$$p_{non} = (\sum_{i=1}^{n}\sum_{j=1}^{m} M \cdot (\sum_{k=1}^{N} p_k/N)^{d_{ij}})/(m \cdot n) \quad (5)$$

We introduce a method to evaluate a data object's popularity below. Different types of applications are always featured by the popular time periods of data objects. For example, the videos in social network applications are usually popular for 3 months [34], while the news in a news website usually is popular for several days. We rank the applications based on their types and use b_{a_i} to denote the rank; a higher b_{a_i} means that the application has longer popular time periods of data objects. Assume time is split into epochs (a fixed period of time). To avoid creating and deleting replicas for data objects that are popular for a short time period, we consider both its application rank (b_{a_i}) and its expected visit frequency, i.e., the number of visits in an epoch (v_{ij}) [16], [19], [35], [36], to determine the popularity function of a data object ($\varphi(\cdot)$):

$$\varphi_{ij}(\cdot) = \alpha \cdot b_{a_i} + \beta \cdot v_{ij} \quad (6)$$

where α and β are the weights for the two factors.

Let r_{ij} be the probability of requesting data D_{ij}. For single-object requests, $\sum_{i=1}^{n}\sum_{j=1}^{m} r_{ij} = 1$ after normalization. The request probability of a data object is the same as that of each partition of this data object. The request probability is proportional to the popularity of the data object ($\varphi(\cdot)$) [8], [37].

$$r_{ij} = k_1 \cdot \varphi_{ij}(\cdot) \quad (7)$$

where k_1 is a certain coefficient.

3) Availability Maximization Problem Formulation: In a cloud system with the co-existence of correlated and non-correlated (uniform and nonuniform) machine failures, the data is lost if any type of failures occurs. Recall that p_{cor} (Formula (2)), p_{uni} (Formula (4)) and p_{non} (Formula (5)) are probabilities of a data object loss caused by correlated machine failures, uniform machine failures, and nonuniform machine failures (at epoch t), respectively. So the probability of data loss caused by correlated and non-correlated machine failures is

$$P_f = w_1 \cdot p_{cor} + w_2 \cdot p_{uni} + w_3 \cdot p_{non} \quad (\sum_{i=1}^{3} w_i = 1) \quad (8)$$

where w_1, w_2, and w_3 are the probabilities of the occurrence of each type of machine failures, respectively.

We maximize the data availability by minimizing the expected request failure probability with the consideration that different data objects have different popularities. To achieve this goal, we present a NLIP model with multiple constraints to determine the replication degree of each data object.

Given the popularities and sizes of data objects, our goal is to find the optimal replication degree of each data object such that the expected request failure probability (denoted by $\bar{P}_r$) can be minimized. Each application purchases a certain storage space in public clouds. Also, in private clouds, different applications may be assigned different storage spaces based on their priorities. Thus, the storage space constraint for each application is necessary and important due to the limited precious storage media. Thus, we formalize our problem as the following constrained optimization problem:

$$\begin{aligned} min \quad & \bar{P}_r = \sum_{i=1}^{n}\sum_{j=1}^{m} r_{ij} \cdot M \cdot (P_f)^{d_{ij}} \\ s.t. \quad & \sum_{j=1}^{m} s_{ij} \cdot d_{ij} \le K_i \;\; (i = 1, ..., n) \end{aligned} \tag{9}$$

where $K_i \;\; (i = 1, ..., n)$ is the space capacity for application i, and s_{ij} is the size of data object D_{ij}. $\sum_{j=1}^{m} s_{ij} \cdot d_{ij}$ is the total storage consumption of all the data objects belonging to application i. The optimization objective is to minimize the expected request failure probability and the constraint is to ensure that the storage consumption of an application does not exceed its space capacity.

B. Consistency Maintenance Cost Minimization

In this section, we formulate the problem of minimizing consistency maintenance cost caused by data replication. We use C_c to denote the total consistency maintenance cost of all replicas in the cloud storage system. We first introduce how to calculate C_c. Previous work [38] indicates that the data object write overhead is linear with the number of data object replicas. Then, the consistency maintenance cost of a partition can be approximated as the product of the number of replicas of the partition and the average communication cost parameter (denoted by δ_{com}) [19], i.e., $d_{ij} \cdot \delta_{com}$. Hence, the total consistency maintenance cost of all data objects is

$$C_c = \sum_{i=1}^{n}\sum_{j=1}^{m} (M \cdot d_{ij}) \cdot \delta_{com} \tag{10}$$

The fixed average communication cost (δ_{com}) can be calculated as in [39]:

$$\delta_{com} = E[\sum_{i,j} s^u \cdot dis(S_i, S_j) \cdot \sigma] \tag{11}$$

where s^u is the average update message size, $dis(S_i, S_j)$ is the geographic distance between the server of the original copy S_i and a replica server S_j (an expectation of all possible distances between server of original copy (primary server) and replica servers, which is calculated it from a probabilistic perspective), and σ is the average communication cost per unit of distance. We adopt the method in [19] to calculate the geographic distance between servers. Specifically, it uses a 6-bit number to represent the distance between servers. Each bit corresponds to the location part of a server, i.e., continent, country, datacenter, room, rack and server. Starting with the most significant bit (i.e., the leftmost bit), each location part of both servers are compared one by one to compute the geo-similarity between them. The corresponding bit is set to 1 if the location parts are equivalent, otherwise it is set to 0. Once a bit is set to 0, all less significant bits are automatically set to 0. For example, suppose S_i and S_j are two arbitrary servers, and the distance between them is represented as 111000 (as shown below). Then, 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. That is,

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

Thus, the optimization problem for consistency maintenance cost can be formulated as follows:

$$\begin{aligned} min \quad & C_c = \sum_{i=1}^{n}\sum_{j=1}^{m} (M \cdot d_{ij}) \cdot \delta_{com} \\ s.t. \quad & \sum_{j=1}^{m} s_{ij} \cdot d_{ij} \le K_i \;\; (i = 1, ..., n) \end{aligned} \tag{12}$$

C. Storage Cost Minimization

Different applications require different storage media (e.g., disk, SSD, EBS) to store data. Different media have different costs per unit size. For example, in the Amazon web cloud service, the prices for EBS and SSD are \$0.044 and \$0.070 per hour, respectively. After satisfying the applications' storage requirements on different storage media, we need to decide the storage media for their additional replicas for enhanced data availability. Different applications have different SLAs with different associated penalties. The applications with higher SLA violation penalties should have higher priorities to meet their SLA requirements in order to reduce the associated penalties. Therefore, the additional replicas of data objects of higher-priority applications should be stored in a faster storage medium. On the other hand, in order to save storage cost, the additional replicas of data objects of lower-priority applications should be stored in a lower and less expensive storage medium. We use b_{p_i} to denote application i's priority; higher b_{p_i} means higher priority. Since faster storage mediums are more costly, for data objects of high priority applications, we hope to store more data partitions in faster storage mediums in order to satisfy more requests per unit time hence increase data availability [19]. Thus, to determine the storage medium for storing additional replicas, we define a priority function $\psi(\cdot)$ of a data object based on its application priority and size [16]:

$$\psi_{ij}(\cdot) = \gamma \cdot b_{p_i} + \eta / s_{ij} \tag{13}$$

where γ and η are the weights for application priority and data object's size. The size of the data object can be changed due to write operation. The priority values are classified to a number of levels, and each level corresponds to a storage medium. Thus, the unit storage cost of a data object equals the unit cost of its mapped storage medium, denoted by $c_{s_{ij}}$, which is proportional to the priority value of a data object ($\psi_{ij}(\cdot)$).

$$c_{s_{ij}} = k_2 \cdot \psi_{ij}(\cdot) \tag{14}$$

where k_2 is a certain coefficient.

The storage cost of a data object is related to its storage medium, its size and the number of its replicas. Different storage mediums have different unit costs, and different data objects have different sizes and replication degrees. We minimize storage cost by minimizing the expected cost for storage mediums. We examine expected storage cost minimization by determining the replication degree for each data object. To achieve this goal, we present a NLIP approach with multiple constraints that can be used to obtain a policy.

Given the applications of data objects and the data object sizes, our goal is to find the replication degree for each data object that minimizes storage cost. Hence, we formalize our problem as the following constrained optimization problem:

$$\begin{aligned} min \quad & C_s = \sum_{i=1}^{n}\sum_{j=1}^{m} (s_{ij} \cdot d_{ij} \cdot c_{s_{ij}} \cdot T) \\ s.t. \quad & \sum_{j=1}^{m} s_{ij} \cdot d_{ij} \le K_i \;\; (i = 1, ..., n) \end{aligned} \tag{15}$$

where T is the time duration of an epoch.

D. Problem Formulation and Solution

We consider additional three threshold constraints for request failure probability (P_r^{th}), consistency maintenance cost (C_c^{th}) and storage cost (C_s^{th}). The probability of expected request failure must be no larger than a threshold P_r^{th}. This constraint is used to ensure that the expected request failure probability is not beyond a threshold, which serves the goal of achieving high data availability. The constraint on the consistency maintenance cost is to ensure that the consistency maintenance cost in one epoch is no larger than a threshold, C_c^{th}. The storage cost in one epoch is no more than threshold C_s^{th}. The constraints on both consistency maintenance cost and storage cost are to ensure that the cost caused by replication is at a low level, which makes the system more efficient and economical. By combining Formulas (9), (12) and (15) and the additional constraints, we can build a NLIP model for the global optimization problem as follows:

$$\begin{aligned} min \quad \{\bar{P}_r + C_c + C_s\} &= \sum_{i=1}^{n}\sum_{j=1}^{m}(r_{ij} \cdot M \cdot (P_f)^{d_{ij}}) \\ &+ \sum_{i=1}^{n}\sum_{j=1}^{m}(M \cdot d_{ij}) \cdot \delta_{com} + \sum_{i=1}^{n}\sum_{j=1}^{m}(s_{ij} \cdot d_{ij} \cdot c_{s_{ij}} \cdot T) \end{aligned} \tag{16}$$

$$s.t. \quad \sum_{j=1}^{m} s_{ij} \cdot d_{ij} \leq K_i \ \ (i = 1,...,n) \tag{17}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{m} r_{ij} \cdot M \cdot (P_f)^{d_{ij}} \leq P_r^{th} \ \ (0 < r_{ij} < 1) \tag{18}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{m}(M \cdot d_{ij}) \cdot \delta_{com} \leq C_c^{th} \tag{19}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{m}(s_{ij} \cdot d_{ij} \cdot c_{s_{ij}} \cdot T) \leq C_s^{th} \tag{20}$$

The decision variables are data objects's replication degrees, and they must be positive integers in practice. The objective is to minimize the request failure probability, the consistency maintenance cost and storage cost. The optimization constraints are used to ensure that the space capacity of data objects belonging to each application is not exceeded, the probability of expected request failure is no more than a threshold P_r^{th}, the consistency maintenance cost and storage cost in one epoch are no more than thresholds C_c^{th} and C_s^{th}, respectively.

Theorem 3.1. *The relaxed NLIP optimization model is convex.*

Proof Inequs. (17), (19), (20) are linear inequalities, and they define convex regions. The exponential function $P_f^{d_{ij}}$ in Inequ. (18) is convex, and the sum (i.e., $\sum_{i=1}^{n}\sum_{j=1}^{m} r_{ij} \cdot m \cdot (P_f)^{d_{ij}}$) of convex functions is a convex function. Thus, the constraint (18) defines a convex set. All the constraints define convex regions, and the intersection of convex sets is a convex set. Thus, the region of the optimization problem is convex. Hence the relaxed NLIP optimization problem is convex. ■

For analytical tractability, we first relax the problem to a real-number optimization problem in which $d_{11},...,d_{nm}$ are real numbers, and derive the solution for the real-number optimization problem. Then, we use integer rounding to get the solution for practical use. Specifically, we adopt the approach from [8] to round each d_{ij} to its nearest integer while all d_{ij}'s smaller than 1 are rounded to 1. By relaxing the problem to real-number optimization problem, the optimal solution should always use up all the available storage space (i.e., K_i) for each application [8].[3] Thus, we have

$$\sum_{j=1}^{m} s_{ij} \cdot d_{ij} = K_i \quad (i = 1,...,n) \tag{21}$$

where K_i is the space capacity for application i. For the real-number optimization problem, we use Lagrange multipliers to derive the solution. Since there are $n+3$ constraints, we use the multipliers $\lambda_1, \lambda_2, ..., \lambda_{n+3}$ to combine the constraints and the optimization goal together into the Lagrangian function

$$\begin{aligned} &\Lambda(d_{11},...,d_{1m},...,d_{n1},...,d_{nm},\lambda_1,\lambda_2,...,\lambda_{n+3}) \\ &= \sum_{i=1}^{n}(\sum_{j=1}^{m} r_{ij} \cdot M \cdot (P_f)^{d_{ij}} + \sum_{j=1}^{m}(M \cdot d_{ij}) \cdot \delta_{com} + \sum_{j=1}^{m}(s_{ij} \cdot d_{ij} \cdot c_{s_{ij}} \cdot T) \\ &+ \lambda_i(\sum_{j=1}^{m} s_{ij} \cdot d_{ij} - K_i)) + \lambda_{n+1}(\sum_{i=1}^{n}\sum_{j=1}^{m} r_{ij} \cdot M \cdot (P_f)^{d_{ij}} - P_r^{th}) + \lambda_{n+2} \\ &(\sum_{i=1}^{n}\sum_{j=1}^{m}(M \cdot d_{ij})\delta_{com} - C_c^{th}) + \lambda_{n+3}(\sum_{i=1}^{n}\sum_{j=1}^{m}(s_{ij} \cdot d_{ij} \cdot c_{s_{ij}} \cdot T) - C_s^{th}) \end{aligned} \tag{22}$$

where $K_i = \sum_{j=1}^{m} s_{ij} \cdot d_{ij}$. The critical value of Λ is achieved only if its gradient is zero. Based on Theorem 3.1, the NLIP optimization problem is convex. Thus, the gap between the relaxed problem and its dual problem is zero [40]. Also, the object function of the NLIP model is derivable, thus the gradients of λ exist. We can get the solution for the optimization problem based on the Lagrange dual solution. Considering that the popularities and importance of data objects usually do not change much within a short period of time, we can choose larger value for T to avoid computation overhead, and also use IPOPT [41], [42] (a software library for large scale nonlinear optimization) to solve the large-scale nonlinear optimization problem, which make MRR more practical.[4]

IV. The MRR Replication Scheme

In Section III, we calculate the replication degree of each data object, which is the first step of the design of MRR. The next problem is how to assign the replicas to nodes, which is of importance for increasing the data availability [9]. Recall that in Section III-A, we introduced BIBD-based file replication in Copyset Replication that can reduce data loss probability in correlated machine failures. Though the BIBD-based method can be used to reduce data loss probability in correlated machine failures, it requires a constant replication degree and cannot be used for replicating data with various replication degrees. Our proposed MRR can deal with the problems. We present the details of MRR below.

Algorithm 1 shows the pseudocode of the MRR replication algorithm. For particular and arbitrary i and j, the replication degree d_{ij} of data D_{ij} can be obtained from the NLIP optimization model in Section III. To reduce data loss in both correlated and non-correlated machine failures, MRR first ranks these replication degrees in ascending order (i.e., $d_1,...,d_l$). For a given replication degree d_i $(i = 1,...,l)$, MRR first groups the data objects with replication degree d_i together (denoted by

[3]The storage cost may increase as storage space increases, but it also depends on the storage media for storing the data, thus the claim that the space constraint holds at equality does not contradict the main hypothesis (cost-effective) of the paper.

[4]Although some parameters (e.g., data popularity, machine failure rate) are not usually available a priori, we can get them from the historical data [7], [8], [16].

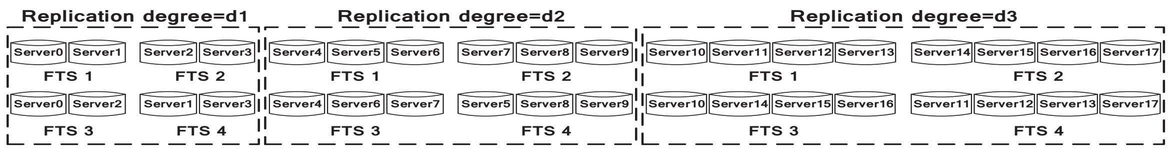


Figure 2: Architecture of MRR for cloud storage fault-tolerant sets (FTSs).

Table I: Parameters from publicly available data [12].

System	Chunks per node	Cluster size	Scatter width
Facebook	10000	1000-5000	10
HDFS	10000	100-10000	200
RAMCloud	8000	100-10000	N-1

D_i), and counts the number of data objects with replication degree d_i (denoted by $N^r_{D_i}$). To handle the problem of varying replication degrees, MRR partitions all nodes to l groups and then conducts Copyset Replication [12] to assign chunks with replication degree d_i to the i^{th} node group $(i = 1, ..., l)$.

For load balance, the number of nodes in each group is

Algorithm 1: Pseudocode of the MRR algorithm.

```
1 Compute the replication degree for each data object using the NLIP
  model (Section III)
2 Rank the replication degrees in ascending order $d_1, ..., d_l$
3 for $i \leftarrow 1$ to $l$ do
4     Group data objects with $d_i$ together ($D_i$)
5     Use BIBD-based method to generate FTSs; each FTS has nodes
      from different datacenters but within a certain geographic distance
6     Store each chunk's replicas of data objects with $d_i$ to all nodes in
      an FTS with $d_i$
7 return
```

proportional to the number of replicas that will be stored in the group. Accordingly, MRR assigns $\lfloor N \cdot \frac{N^r_{D_i} \cdot d_i}{\sum_{i=1}^{l} N^r_{D_i} \cdot d_i} \rfloor$ nodes to data group D_i. MRR then uses the BIBD-based method to generate FTSs for each group of nodes. Specifically, MRR determines λ in Section III-A1 based on the load balancing requirement and then uses the BIBD-based method for (N, R, λ), where N is the number of the nodes in a node group and R is the replication degree of the group (d_i) (R relates the correlated machine failure with replication degree). The nodes in each FTS are required to be from different datacenters and within a certain geographic distance between each other. Distributing replicas over different datacenters can avoid data loss due to machine failures (e.g., caused by power outages) for data reliability. Limiting the distances between replica nodes for a data chunk constrains the consistency maintenance cost. MRR replicates the chunks of each data object to all nodes in an FTS.[5] Figure 2 illustrates an example to show the design of MRR. In the example, $d_1 = 2$, $d_2 = 3$, and $d_3 = 4$. In the left most block, each FTS contains two chunks of a data object. Each FTS overlaps with every other FTS with at most one server. This helps reduce the probability of data loss in correlated machine failures and balance the load. For those data objects with replication degree 2, the chunks of a data object will be stored to the nodes in an FTS in the left most block. In the middle block, each FTS contains three chunks of a data object. In the right most block, each FTS contains four chunks of a data object.

[5]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 the probability of data loss caused by correlated machine failures using BIBD-based method [12].

Table II: Parameter settings.

Parameter	Meaning	Setting
N	# of servers	1000-10000
M	# of chunks of a data object	3 [45]
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
P_r^{th}	Threshold for expected request failure	0.05
C_c^{th}	Threshold for consistency maint. cost	1000000
C_s^{th}	Threshold for storage cost	300000
m	# of data objects in each application	1000
n	# of data applications	5

Recall that each data priority value corresponds to a storage medium. We assume that all servers have all types of storage media, and we need to determine which medium to use for a given priority on a given server. When replicating a chunk to a node, MRR chooses the storage media for the chunks based on data objects' priority calculated by Formula (13). Data objects with higher priority values will be stored to faster and more expensive storage media (e.g., Memory, SSD), and vice versa. The constraint (17) is to ensure that storage requirements of data objects do not overfill the servers.

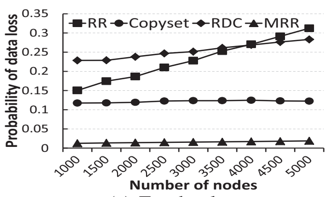
V. Performance Evaluation

We conducted the numerical analysis based on the parameters in [12] (Table I) derived from the system statistics from Facebook, HDFS and RAMCloud [1], [2], [10], [12], [13], [43], and also conducted real-world experiments on Amazon S3. The distributions of file read and write rates in our analysis and tests follow those of the CTH trace [44] provided by Sandia National Laboratories that records 4-hour read/write log in a parallel file system.

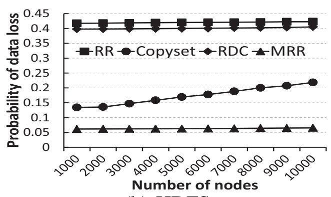
A. Numerical Analysis Results

We compared MRR with other three replication schemes: Random Replication (RR) [12], Replication Degree Customization (RDC) [8] and Copyset Replication [12] (Copyset). We compare MRR with Copyset instead of Tiered Replication (TR) [29] because TR focuses on data durability [29], however MRR focus on data availability. Moreover, the key idea of Copyset and TR for reducing data loss probability is the same, and they both use the BIBD-based method to reduce the probability of data loss. RR places the primary replica on a random node (say node i) in the entire system, and places the secondary replicas on (R-1) nodes around the primary node (i.e., nodes i+1, i+2,...).[6] RDC derives replication degree for each object with considering data object popularity to maximize the expected data availability for non-correlated machine failures. 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

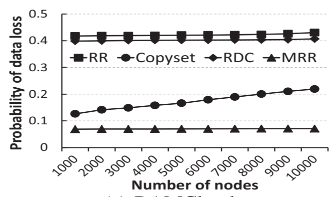
[6]RR is based on Facebook's design, which chooses secondary replica holders from a window of nodes around the primary node.



(a) Facebook

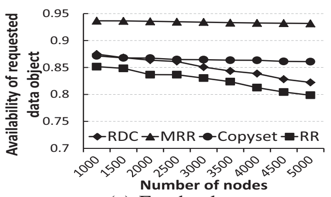


(b) HDFS

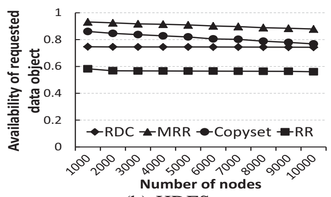


(c) RAMCloud

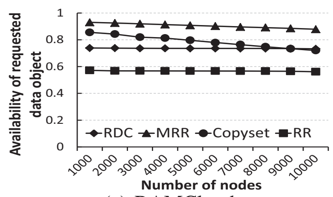
Figure 3: Probability of data loss vs. number of nodes ($R = 3$ for RR and Copyset).



(a) Facebook

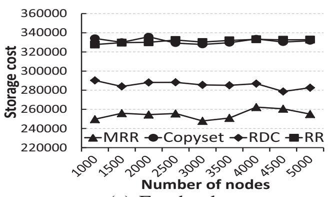


(b) HDFS

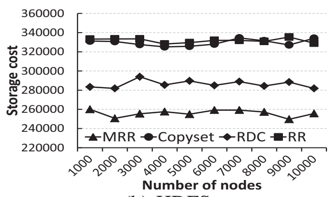


(c) RAMCloud

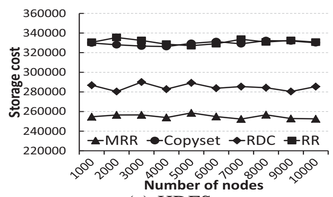
Figure 4: Availability of requested data objects vs. number of nodes ($R = 3$ for RR and Copyset).



(a) Facebook

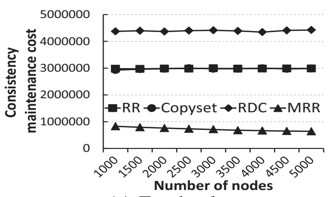


(b) HDFS

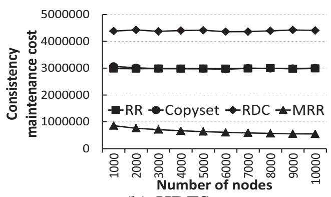


(c) HDFS

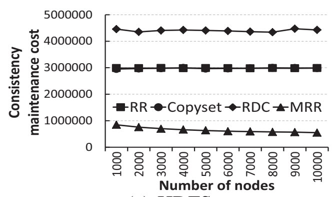
Figure 5: Storage cost vs. number of nodes ($R = 3$ for RR and Copyset).



(a) Facebook



(b) HDFS



(c) HDFS

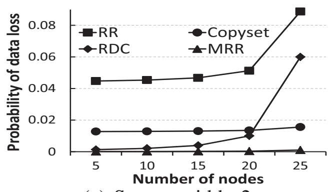
Figure 6: Consistency maintenance cost vs. number of nodes ($R = 3$ for RR and Copyset).

data loss by minimizing the number of copysets. The number of nodes that fail concurrently in each system was set to 1% of the nodes in the system [6], [12]. Since this rate is the maximum percentage of concurrent failure nodes in real clouds (i.e., worst case) [10], [11], [46], it is reasonable to see higher probabilities of data loss and lower expected data availability in our analytical results than the real results in current clouds. The distributions of file popularity and updates follow those of the CTH trace. We used the normal distribution with mean of 10 and standard deviation of 1 to generate 10 unit costs for different storage mediums. Compared to uniform machine failures, nonuniform and correlated machine failures are more realistic due to different hardware/software compositions and configurations [8], [47]. Thus, we set $w_1 = 0.4$, $w_2 = 0.2$ and $w_3 = 0.4$ in Equ. (8), respectively. We randomly generated 6 bit number from reasonable ranges for each node to represent its location, as explained in Section III-B. Table II shows the parameter settings in our analysis unless otherwise specified.

We first calculate the data loss probability for each method.[7] Specifically, we used Formula (8) for MRR, Formula (2) for Copyset, Formula (3) for RR, and Equ. (8) with $w_1 = 0$, $w_2 = 0.4$ and $w_3 = 0.6$ for RDC. Figure 3(a)-3(c) show the relationship between the probability of data loss and the number of nodes in the Facebook, HDFS and RAMCloud environments, respectively. We find that the result approximately follows MRR<Copyset<RDC<RR. The probability of data loss in Copyset is higher than that in MRR. This is because MRR considers non-correlated machine failures which are not considered in Copyset. RDC generates a higher probability of data loss than MRR and Copyset because it neglects reducing the probability of data loss caused by correlated machine failures. The probability of data loss in RR is much higher than MRR and Copyset, and also higher than RDC. This is due to two reasons. First, RR places the copies of a chunk on a certain number (i.e., R) of different nodes. Any combination of R nodes that fail simultaneously would result in data loss in correlated machine failures. However, MRR and Copyset lose data only if all the nodes in an FTS fail simultaneously. Second, MRR and RDC consider data popularity to increase the expected data availability, which is, however, not considered in RR.

We then calculate the availability of requested data object by $1 - \bar{P}_r$, and $\bar{P}_r$ is calculated by Formula (9). Figure 4(a)- 4(c) show the result of the availability of requested data objects. We see the result generally follows MRR>Copyset>RDC>RR in all figures. The availability of requested data objects in Copyset is lower than that in MRR because MRR considers

[7]Many datacenter operators prefer to low probability of any incurring data loss at the expense of losing more data in each event due to high cost of each incident of data loss [12].


(a) Scatter width=2

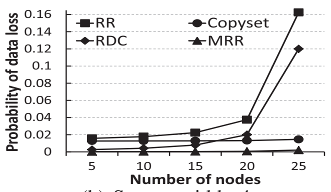
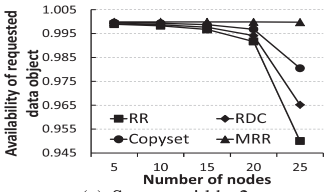

(b) Scatter width=4

Figure 7: Probability of data loss vs. number of nodes on Amazon S3 ($R = 3$ for RR and Copyset).


(a) Scatter width=2

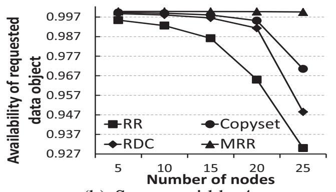
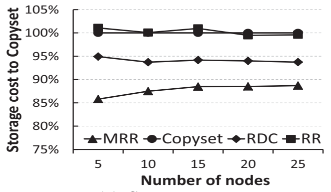

(b) Scatter width=4

Figure 8: Availability of requested data objects vs. number of nodes on Amazon S3 ($R = 3$ for RR and Copyset).


(a) Storage cost

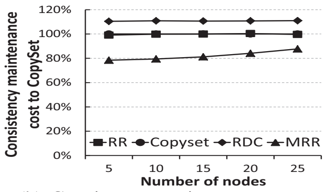

(b) Consistency maintenance cost

Figure 9: Storage cost and consistency maintenance cost vs. number of nodes on Amazon S3 ($R = 3$ for RR and Copyset).

data popularity when determining the replication degree for each chunk, which is, however, not considered in Copyset. Also, MRR reduces data loss in both correlated and non-correlated machine failures, while Copyset only minimizes the data loss in correlated machine failures. The availability of requested data objects in Copyset is higher than that in RDC because RDC cannot reduce the probability of data loss caused by correlated machine failures and thereby decreases the availability of requested data objects. RR has the lowest availability because RR places the copies of a chunk on a certain number (i.e., R) of nodes and any combination of R nodes that fail simultaneously would cause data loss. Also, RR does not consider data popularity as RDC.

We then used Formula (15) to calculate the storage cost based on the sizes, replication degrees, and storage medium unit costs of data objects for MRR. For the other three methods, we randomly choose storage media for data objects and do not minimize the storage cost with space capacity constraint (Inequ. 17) for RDC. Figure 5(a)-5(c) show the result of storage cost. The result in all three figures generally follows RR≈Copyset>RDC>MRR. This is because MRR stores data objects into different storage mediums based on their applications' priorities, the sizes, and the replication degrees of data objects to minimize the total storage cost. The storage costs in Copyset and RR are higher than RDC and much higher than MRR. RDC reduces the replicas of unpopular data objects, thus reducing storage cost. Copyset and RR neither consider the different storage mediums nor reduce the replicas of unpopular data objects to reduce storage cost.

We used Formula (12) to calculate the consistency maintenance cost of each method based on the geographic distance and replication degrees of data objects. Figure 6(a)-6(c) show the result of consistency maintenance cost. In these figures, the consistency maintenance costs in Copyset and RR are higher than MRR because MRR limits the geographic distance between the replica nodes of a chunk and the number of replicas, thereby reducing the consistency maintenance cost. However, Copyset and RR neglect geographic distances. RDC produces the highest consistency maintenance cost because RDC neglects geographic distance and it also generates more replicas for popular data objects.

B. Real-world Experimental Results

We conducted experiments on the real-world Amazon S3. We simulated the geo-distributed storage datacenters using three regions of Amazon S3 in the U.S. In each region, we created the same number of buckets, each of which simulates a data server. The number of buckets was varied from 5 to 25 with step size of 5 in our test. We distributed 5000 data objects to all the servers in the system. We used the distributions of read and write rates from the CTH trace data 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 (unavailable data objects).[8] We used the parameters in Table II except p and N. In this test, N is the total number of simulated data servers in the system and p (with average value 0.089) follows the actual probability of a server failure in the real system. We used the actual price of the data access of Amazon S3 to calculate the storage cost.

Figure 7(a) and 7(b) show the result of probability of data loss on Amazon S3. We see the result approximately follows MRR<RDC<Copyset<RR. Our numerical result shows that MRR<Copyset<RDC<RR. Both results confirm that MRR generates the lowest probability of data loss. RDC generates higher probability of data loss than Copyset in the numerical analysis but generates lower probability of data loss than Copyset in the experiments. This is because RDC cannot handle correlated machine failures, and the failure rate of correlated machine failures is 1% in the numerical analysis but our real-world experiment has fewer correlated machine failures. We also see that scatter width 2 produces lower probability of data loss than scatter width 4. This is because a large scatter width increases the number of FTSs and thus increases the probability of data loss.

Figure 8(a) and 8(b) show result of the availability of requested data objects on Amazon S3. We see the result follows MRR>Copyset>RDC>RR, which is consistent with the result in Figure 4 due to the same reasons. We also see that scatter width 2 produces higher availability than scatter width 4 because a large scatter width increases the probability of data loss and reduces data availability.

We then regard Copyset as a baseline and calculate the ratio of storage/consistency maintenance cost of each of the other

[8]Since it is hard to generate permanent failures on Amazon S3 and the network latency is low based on [48], we believe the data request failure within U.S. is mainly caused by machine failures, and we consider the requests targeting each region with latency longer than 100ms as failed request, which reflects the availability of data objects.

methods over Copyset. Figure 9(a) and 9(b) show the storage cost ratio and consistency maintenance cost ratio of different schemes, respectively. We see the storage cost ratio follows RR≈Copyset>RDC>MRR, and the consistency maintenance cost ratio follows RDC>RR≈Copyset>MRR, which are consistent with that in Figure 5 and 6 due to the same reasons.

VI. Conclusion

In this paper, in order to increase data availability and reduce cost caused by replication, we formulate a problem that determines the replication degree of each data object so that the request failure probability, consistency maintenance cost and storage cost are minimized in a cloud storage in both correlated and non-correlated machine failures. Based on the problem solution, we propose the MRR scheme that assigns the chunk replicas of data objects to the nodes to handle the aforementioned problems for the objective. Our extensive numerical analysis and real-word experiments on Amazon S3 show that MRR outperforms other replication schemes in different performance metrics. In the future, we will further consider data update frequency for reducing consistency maintenance cost, the effects of node joining and leaving, and the influence of changing network connections. Also we will consider energy consumption of machines and designing an optimal replication scheme to save power.

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] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In *MSST*, pages 1–10, 2010.

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

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

[4] 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.

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

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

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

[9] A. Haeberlen, A. Mislove, and P. Druschel. Glacier: Highly durable, decentralized storage despite massive correlated failures. In *NSDI*, 2005.

[10] R. Chansler. Data availability and durability with the hadoop distributed file system. *;login: The USENIX Magazine*, 37(1), 2012.

[11] J. Dean. Evolution and future directions of large-scale storage and computation systems at google. In *Proc. of SoCC*, page 1, 2010.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

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

[17] E. Keller, J. Szefer, J. Rexford, and R. B. Lee. Nohype: Virtualized cloud infrastructure without the virtualization. In *Proc. of ISCA*, 2010.

[18] 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.

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

[20] H. Yu and A. Vahdat. The costs and limits of availability for replicated services. In *Proc. of SOSP*, pages 29–42, 2001.

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

[22] C. Tang and S. Dwarkadas. Hybrid global-local indexing for efficient peer-to-peer information retrieval. In *Proc. of NSDI*, 2004.

[23] 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.

[24] Y. Ma, T. Nandagopal, K. P. N. Puttaswamy, and S. Banerjee. An ensemble of replication and erasure codes for cloud file systems. In *Proc. of INFOCOM*, pages 1276–1284, 2013.

[25] A. Thomson and D. Abadi. Calvinfs: Consistent wan replication and scalable metadata management for distributed file systems. In *Proc. of FAST*, Santa Clara.

[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*, Indianapolis, 2010.

[28] Amazon S3. http://aws.amazon.com/s3 [accessed in Jun. 2016].

[29] 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*, pages 31–43, 2015.

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

[31] H. Yu, P. Gibbons, and S. Nath. Availability of multi-object operations. In *Proc. of NSDI*, pages 211–224, 2006.

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

[33] P. Kaski and P. Östergård. There exists no (15, 5, 4) rbibd. *Journal of Combinatorial Designs*, 9:227–232, 2001.

[34] H. Shen, Z. Li, Y. Lin, and J. Li. SocialTube: P2P-assisted Video Sharing in Online Social Networks. *TPDS*, 2014.

[35] 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.

[36] J. Liu, H. Shen, and L. Yu. Question quality analysis and prediction in community question answering services with coupled mutual reinforcement. *TSC*, PP(99):1–14, 2015.

[37] H. Shen, Z. Li, J. Liu, and J. E. Grant. Knowledge sharing in the online social network of yahoo! answers and its implications. *IEEE Transactions on Computers (TC)*, 64(6):1715C1728, June 2015.

[38] 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.

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

[40] M. Bazaraa, H. Sherali, and C. Shetty. Nonlinear programming: Theory and algorithms. *Wiley Interscience*, 2006.

[41] IPOPT. https://projects.coin-or.org/Ipopt [accessed in Jun. 2016].

[42] A. Wächter and L.T. Biegler. On the implementation of an interior-point filter linesearch algorithm for large-scale nonlinear programming. *Math. Program*, 106(1), 2006.

[43] Intelligent block placement policy to decrease probability of block loss. https://issues.apache.org/jira/browse/HDFS-1094.

[44] N. Nakka, A. Choudhary, R. Klundt, M. Weston, and L. Ward. Detailed analysis of i/o traces for large scale applications. In *HiPC*, 2009.

[45] J. Cook, A. Wolf, and B. Zorn. Partition selection policies in object database garbage collection. In *Proc. of SIGMOD*, New York, 1994.

[46] 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*, Berkeley, CA, USA, 2010.

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

[48] S3 FAQs. https://aws.amazon.com/s3/faqs [accessed in Jun. 2016].