A Low-Cost Multi-Failure Resilient Replication Scheme for High Data Availability in Cloud Storage

Jinwei Liu, *Member, IEEE, ACM,* Haiying Shen*, *Senior Member, IEEE, Member, ACM, Hongmei Chi, Husnu S. Narman, Yongyi Yang, Long Cheng, and Wingyan Chung*

***Abstract*—Data availability is one of the most important performance factors in cloud storage systems. To enhance data availability, replication is a common approach to handle the machine failures. However, previously proposed replication schemes cannot effectively handle both correlated and non-correlated machine failures, especially while increasing the data availability with limited resources. The schemes for correlated machine failures must create a constant number of replicas for each data object, which often neglects diverse data popularities and does not 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 costs 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 therefore 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 (correlated and non-correlated machine failures) 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 and storage cost. Extensive numerical results from trace parameters and experiments from real-world Amazon S3 demonstrate that MRR achieves high data availability, low data loss probability and low consistency maintenance and storage costs when compared to previous replication schemes.**

***Index Terms*—Cloud storage, Replication, Data availability, Cost-effectiveness.**

I. Introduction

Cloud Computing provides a large range of services to a number of different types of users. To provide high quality-of-services, datacenter management, network, and fault tolerance must be well designed by the service providers. Several datacenter storage systems such as Hadoop Distributed File System (HDFS) [1], RAMCloud [2], Google File System (GFS) [3] and Windows Azure [4] have been built because datacenter storage system 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 are executed within 300ms [5]. Therefore, a storage system must guarantee data availability for different applications to comply with SLAs, which is a non-trivial task. In many cloud services such as Amazon, Google App Engine and Windows Azure, the server failure probability is typically in the range of [0.01%, 10%] [6]–[9] and the data availability is usually in the range of [99.9%, 99.99%] [10]. The data access failures that lead to SLA violations and associated penalties, degrade the quality-of-service of applications and also may lead to the loss of cloud customers, that in turn, could lead to significant losses for the companies. For example, Amazon's one second delay of data availability can cost them $1.6 billion, and Google could potentially lose eight million searches per day because of a 0.4 second delay [11].

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 [12]–[14] 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 [15]–[17], a non-negligible percentage (0.5%-1%) of nodes [1], [15] do not come back to life after power is restored [18]. Such correlated machine failures cause significant data loss [18], which have been documented by Yahoo! [1], LinkedIn [15] and Facebook [19]. Non-correlated machine failures [13] are caused by reasons that include different hardware/software compositions and configurations, and varying network access ability. Measurements of over 10,000 file systems on desktop computers at Microsoft [20] demonstrate that machines experience disk head crashes hence permanent data-loss failures in a temporally uncorrelated fashion. Non-correlated failures can be classified into uniform and nonuniform machine failures, in which machines fail with the same and different probabilities (possibly due to the same or different computer configura-

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

J. Liu and H. Chi are with the Department of Computer and Information Sciences at Florida A&M University, Tallahassee, FL 32307, USA (e-mail: jinwei.liu@famu.edu; hongmei.chi@famu.edu).

H. Shen is with the Computer Science Department at the University of Virginia, Charlottesville, VA 22904, USA (e-mail: hs6ms@virginia.edu).

H. S. Narman is with the Computer Science Department at Marshall University, Huntington, WV 25755, USA (e-mail: narman@marshall.edu).

Y. Yang is with Computer Engineering Department at the University of Virginia, Charlottesville, VA 22904, USA (e-mail: yy4av@virginia.edu).

L. Cheng is with the Insight SFI Research Centre for Data Analytics, School of Computing, Dublin City University, Dublin, Ireland (e-mail: long.cheng@dcu.ie).

W. Chung is with the Institute for Simulation and Training at University of Central Florida, Orlando, FL 32826, USA (e-mail: wchung@ucf.edu).

tions), respectively.

Replication is a common approach to reduce data loss and enhance data availability. Due to the requirements on storage efficiency, the data popularity consideration in replication is critical to maximize the expected data availability[1]. Due to highly skewed data popularity distributions [21], [22], 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) [23] to increase expected data availability.

Although the strategy of creating more replicas for data objects improves data availability, it comes with higher consistency maintenance cost [24], [25] and storage cost [26]. 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 costs, 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 a 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) [25], [27].

Random replication, as a popular replication scheme, 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 random 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 [18]. 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 [12], [13], [17], [18]. Although many methods have been proposed to improve data availability [12], [13], [17], [24], [28], [29], 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.[2] To address the problem, we propose a low-cost multi-failure resilient replication scheme (MRR). MRR outperforms previous replication schemes (e.g., Random Replication, Copyset Replication [18]) 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.

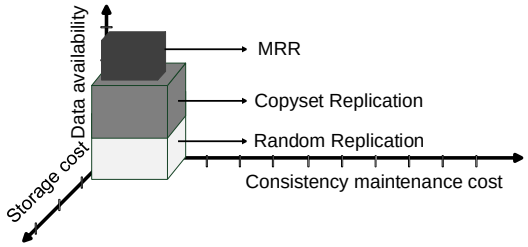


Figure 1: Achieving an optimal tradeoff among data availability, storage cost and consistency maintenance cost.

- 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. We also introduce the concept of the correlation between data objects to analyze the probability of two data objects being requested concurrently or sequentially to further reduce the request delay and replication cost.
- 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 [18]; 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 to compare MRR with other state-of-the-art replication schemes. Results show that MRR achieves high data availability, low data loss probability along with 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 describes the data clustering and mapping. Section VI presents the numerical and experimental results. Section VII 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.* [13] assumed independent machines failures, and proposed a model that achieves high expected service availability. However, this model does not consider correlated machine failures, hence it cannot handle such failures. Nath *et al.* [12] identified a set of design principles that system builders can use to tolerate correlated

[1]We use a service-centric availability metric: the proportion of all successful service requests over all requests [13].

[2]In this paper, we focus on improving data availability while reducing the costs caused by replication instead of advancing the well-known CAP theorem.

failures. Cidon *et al.* [18] proposed Copyset Replication to reduce the frequency of data loss caused by correlated machine failures by limiting the replica nodes of many chunks to a single copyset. Zhai *et al.* [30] proposed Independence-as-a-Service (INDaaS), an architecture to audit the independence of redundant systems proactively, thus avoiding correlated failures. Zhai *et al.* [31] proposed a novel language framework (RepAudit) that manages to prevent correlated failure risks before service outages occur, by allowing cloud administrators to proactively audit the replication deployments of interest. 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, our proposed MRR reduces probability of data loss caused by both correlated and non-correlated machine failures with the consideration of data popularity. 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 by reducing consistence maintenance and storage costs, which is critical for cloud providers to maximize their revenue.

Ford *et al.* [17] analyzed various failure loss scenarios on GFS clusters, and proposed geo-replication as an effective technique to prevent data loss under large scale concurrent node failures. However, they did not provide a specific strategy for reducing data loss caused by both correlated and non-correlated machine failures while minimizing cost (e.g., consistency maintenance cost and storage cost) caused by replication. The previously proposed methods cannot handle both correlated machine failures and non-correlated machine failures while utilizing limited resources to increase data availability. They neglect the data popularity existing in the current cloud storage system and thus could not fully utilize the resource to increase data availability. Also, they do not consider minimizing the consistency maintenance and storage costs caused by replication.

There is a large body of work on enhancing data availability for distributed storage systems. Abu-Libdeh *et al.* [28] presented a replication protocol for datacenter services to provide strong consistency and high availability. Bonvin *et al.* [24] 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. Ford *et al.* [17] characterized the availability properties of cloud storage systems and presented statistical models that enable further insight into the impact of multiple design choices (e.g., data placement and replication strategies). Yu *et al.* [32] 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. Zhang *et al.* [33] proposed a cost-efficient data hosting scheme (CHARM) with high availability in heterogenous multi-cloud. CHARM intelligently puts data into multiple clouds with minimized monetary cost and guaranteed availability. However, most of the previous work neglects data object popularities, particularly when determining the number of replicas for each object, and thus cannot satisfy the demands of popular data objects or fully utilize the limited resource.

Table I: Notations

N	Total # of nodes	C_s^{th}	Upper bound of C_s
T	One epoch duration	φ_{ij}	Popularity function of D_{ij}
p	Prob. of a server failure	ψ_{ij}	Priority function of D_{ij}
D_{ij}	The jth data in app. i	K_i	Space capacity for app. i
s_{ij}	The size of D_{ij}	r_{ij}	Prob. of requesting D_{ij}
d_{ij}	Replication degree of D_{ij}	v_{ij}	# of visits to D_{ij} / epoch
M	# of chunks of each data	$c_{s_{ij}}$	D_{ij}'s unit storage cost
P_f	Prob. of data loss	p_{uni}	Prob. of data loss in uniform machine failures
a_i	app. i		
C_s	Total storage cost	p_{cor}	Prob. of data loss in correlated machine failures
C_c	Consistency maint. cost		
S	Scatter width	p_{non}	Prob. of data loss in non-uniform machine failures
n	# of applications		
C_c^{th}	Upper bound for C_c	T_{orig}	Request time w/o clustering
m	# of data objects / app.	$\bar{P}_r$	Expected request failure prob.
b_{a_i}	App. rank	R_τ	Corr. b/w two data objects
δ_{com}	Ave. comm. cost	$P_c(\chi_1,\chi_2)$	Concurrent access freq. of χ_1, χ_2
P_r^{th}	Upper bound of $\bar{P}_r$	$P_s(\chi_1,\chi_2)$	Sequential access freq. of χ_1, χ_2

Although the work in [13], [21] 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 [12]. 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 work, we propose MRR that can effectively handle both correlated and non-correlated machine failures and also considers the factors that were discussed in Section I to maximize data availability and reduce the consistency maintenance and storage costs.

III. Nonlinear Integer Programming Model for MRR

A cloud storage system usually serves multiple applications simultaneously. Without the loss of generality, we assume there are n applications in the cloud storage, and each application has m data objects [24]. Each data object belongs to only one application, and is split into M partitions [24] and the data object is lost if any of its partitions are lost [18]. The 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 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" [24]. For easy reference, Table I lists the main notations used in this paper.

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 diverse popularities, and then have dissimilar 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 [18], 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 FTS). In response, 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 [18], 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 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[3].

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:

$$p_{cor} = \frac{S}{R-1}\frac{N}{R}/\binom{N}{R}. \tag{2}$$

In random replication, the number of FTSs created is [18]:

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

Based on Eqs. (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 which is obtained according to Equ. (3) [18]. 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 [34], [35]. 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 [18]. 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 [18].

2) Uniform Machine Failures: In the scenario of uniform machine failures, the failures of machines are statistically independent of each other. Each machine has the same probability to fail, denoted by p_u $(0 < p_u < 1)$. The data object is lost if any chunk (partition) of the data object is lost, and a

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

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:

$$p_{uni} = \sum_{i=1}^{n}\sum_{j=1}^{m}(1-(1-p_u^{d_{ij}})^M)/(m\cdot n), \tag{4}$$

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.

3) Nonuniform Machine Failures: Machines in a storage system may experience nonuniform machine failures due to different hardware/software compositions and configurations. 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 the work in [13], 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,

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

4) Data Object Popularity: 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 [36], 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}) [24], [37], to determine the popularity function of a data object ($\varphi(\cdot)$):

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

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 ($\varphi(\cdot)$).

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

where k_1 is a certain coefficient.

5) 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

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

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:

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

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}) [24], 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 from a probabilistic perspective), and σ is the average communication cost per unit of distance. We adopt the method in [24] 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:

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

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 [24]. 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 [21]:

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

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:

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

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 [40]:

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

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

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

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

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

The decision variables are data objects' 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 prob-

ability 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 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 [13] 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[4] [13]. 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

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

where $K_i = \sum_{j=1}^{m} s_{ij}\cdot d_{ij}$. The critical values 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 [41]. 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 [42], [43] (a software library for large scale nonlinear optimization) to solve the large-scale nonlinear optimization problem, which make MRR more practical.

[4]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.

Table II: Parameters from publicly available data [18].

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

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 [14]. 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 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 [18] to assign chunks with replication degree d_i to the i^{th} node group $(i = 1,...,l)$. For load balance, the number of nodes

Algorithm 1: Pseudocode of the MRR algorithm.

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

in each group is proportional to the number of replicas that will be stored in the group. Accordingly, MRR assigns $|N \cdot \frac{N^r_{D_i}\cdot d_i}{\sum_{i=1}^{l} N^r_{D_i}\cdot d_i}|$ 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 [44], [45]. Limiting the

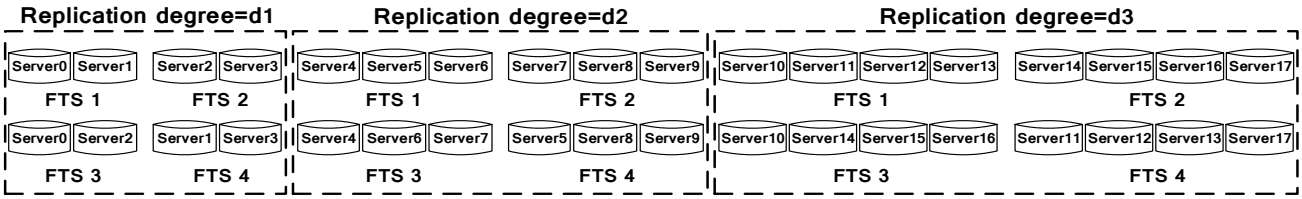


Figure 2: Architecture of MRR for cloud storage fault-tolerant sets (FTSs).

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.

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. Data Objects Clustering and Mapping

In cloud storage system, users may request multiple similar (or correlated) data objects (e.g., files) concurrently or sequentially. It is important to group correlated data objects and store them in closer locations and leverage locality to reduce the request delay and cost (consistency maintenance cost) [46]–[48].

A. Definition of Correlation

In this section, we first introduce the concept of the correlation between data objects, then we leverage the correlation between data objects and propose the data objects clustering and mapping algorithm to further reduce the cost (consistency maintenance cost) and the request delay (thereby improving data availability). The key idea is to store the data objects that have higher possibility to be accessed concurrently or sequentially by leveraging the data correlation, and speed up the data object request or updating request by merging requests on correlated data objects into a single request. The rationale of our algorithm results from locality and cache in computer systems. Denote $P_c(\chi_1, \chi_2)$ and $P_s(\chi_1, \chi_2)$ as the frequencies of two data objects, χ_1 and χ_2 being accessed concurrently and sequentially during a period of time τ, respectively. Then the correlation for χ_1 and χ_2 can be calculated as:

[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 [18].

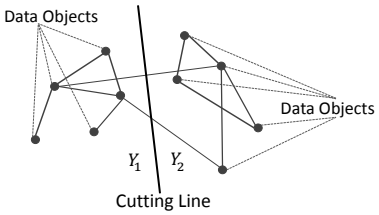


Figure 3: Data objects clustering based on minimum cut tree algorithm.

$$R_\tau = \alpha \cdot P_c(\chi_1, \chi_2) + \beta \cdot P_s(\chi_1, \chi_2) + (1 - \alpha - \beta) \cdot R_{\tau-1}, \tag{23}$$

where α and β are the weights for the frequency of concurrently accessing and sequentially accessing data. $R_{\tau-1}$ is the correlation between χ_1 and χ_2 in the previous time period $\tau - 1$. P_c and P_s are calculated from the historical records of data access, and they can be calculated as

$$\begin{aligned} P_c(\chi_1, \chi_2) &= T_c(\chi_1, \chi_2)/(T_{\chi_1} + T_{\chi_2}) \\ P_s(\chi_1, \chi_2) &= T_s(\chi_1, \chi_2)/(T_{\chi_1} + T_{\chi_2}), \end{aligned} \tag{24}$$

where $T_c(\chi_1, \chi_2)$ is the time in the record that χ_1 and χ_2 were accessed concurrently, $T_s(\chi_1, \chi_2)$ is the time in the record that χ_1 and χ_2 were accessed sequentially. T_{χ_1} and T_{χ_2} are the total access times of χ_1 and χ_2, respectively. α and β in Formula (23) are also calculated from the record based on total concurrent and sequential accesses. Our methodology is an off-line policy since the cost of calculation of R_τ could be high. R_τ for each pair of data objects is calculated dynamically.

B. Data Objects Clustering

After getting the correlation between data objects, we build a graph for m data objects in each application. In the graph, each vertex is a data object and each edge represents the relation between two data objects. We use minimum cut tree algorithm to divide the graph into two parts and allocate all the data objects with higher correlation to closer locations as shown in Figure 3. Below we present a simple method to build the correlation graph and cluster the data. The front-end server receives the request for fetching the data objects and thus be able to calculate the correlation between data objects. Then, the server generates a weighted undirected graph $G(V, E)$, where V is the set of data objects and E is the set of edges connecting data objects. The weight of edge connecting data object χ_1 and χ_2 is the correlation coefficient between them at current time, denoted by $R_T(\chi_1, \chi_2)$. We apply the minimum cut tree algorithm to the graph and cluster data objects into different subsets. Then, we map different subsets of data objects to different servers. As shown in Figure 3, we cut the graph $G(V, E)$ into two clusters Y_1 and Y_2. Each cluster contains a subset of data objects with higher correlation coefficient, denoted by high $R_T(\chi_1, \chi_2)$. The value of the cut equals the sum of the weights of the edges across the cut line. Thus, we have:

$$V^c_{Y_1, Y_2} = \sum_{\chi_1 \in Y_1} \sum_{\chi_2 \in Y_2} R_\tau(\chi_1, \chi_2). \tag{25}$$

The minimum cut tree algorithm achieves the relatively smallest value for the cut as well as large sum of weights inside each class [49]. Therefore, we could map the data objects in one cluster to the same server, and map data objects in different clusters to adjacent servers. However, the key challenge is to guarantee the enough space in each target server to store the clustered data objects when applying this strategy. Algorithm 2 shows the pseudocode of the clustering of data objects and the mapping to servers. Given a graph $G(V,E)$ and a tree in which each tree node represents the free capacity of a data server and the distance between tree nodes is the geographic distance between the servers.

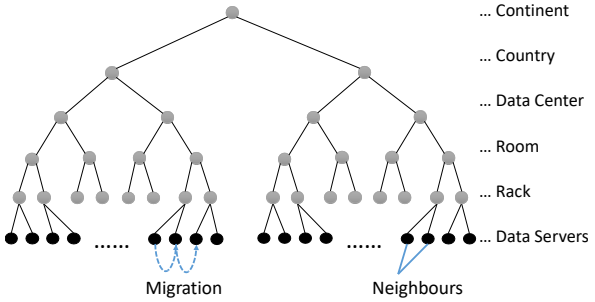


Figure 4: The tree structure of the servers.

In Section III-B, we represent the distance between servers as a 6-bit number. Each bit corresponds to the location part of a server, i.e., continent, country, datacenter, room, rack and server, which means that the maximum depth (denoted by $depth_M$) of the tree structure is six as shown in Figure 4. Since

Algorithm 2: Pseudocode of the data objects clustering and mapping algorithm.

```
Compute the size of the given graph G(V,E), denoted as S_G
Compare the size of data objects in a data cluster and the free
  capacity of data servers S_{D_1}, S_{D_2},...,S_{D_N}
if S_{D_α} is greater than S_G then
    Map the data objects in G(V,E) to data server D_α
    return
for i ← 1 to 2^{depth_M} do
    Compute the cutting line of the given graph G(V,E)
    Compute the size of two clusters Y_1 and Y_2 after cutting
    if exist two data servers D_1 and D_2 with capacity larger than
      Y_1 and Y_2 then
        Choose two data servers D_1^* and D_2^* with smallest distance
        Map Y_1 to D_1^* & map Y_2 to D_2^*
        Break
    else
        Choose the cluster which cannot be mapped, go to step 6
return
```

our algorithm applies to the data objects in one application each time, i.e., the correlation of data is calculated based on the data objects in one application, the depth of six is enough. We first compare the size of data objects in a data cluster and the capacity of data servers, and if there exists a server that can hold the data in the cluster, we map the data to this server. If not, we further partition the graph into two parts, Y_1 and Y_2, to find data servers D_1 and D_2 whose free capacities are larger than the size of data objects in Y_1 and Y_2. To reduce the distance between data objects, we aim to find two clusters which have the nearest distance among the candidates which are D_1^* and D_2^*. Then, we map these two clusters to the corresponding data servers. If the servers cannot meet the requirement of capacity, the algorithm further partitions Y_1 and Y_2 until it finds enough number of servers to store the data or the depth of tree reaches six. It is possible that the algorithm cannot find a subset of servers in which each has enough space to store the data clusters when the depth of the tree has reached $depth_M$. In that case, the algorithm migrates the redundant data objects to adjacent servers who have enough space available and shortest geographic distance to the server where the redundant data objects are migrated.

Table III: Comparison of various replication schemes.

Methods	Computational complexity	Storage overhead	Availability	Data loss rate
RR	$O(\lceil \frac{S}{R-1} \rceil N + Smn)$	$O(mn)$	$O(1)$	$O(1)$
Copyset	$O(\lceil \frac{S}{R-1} \rceil N + \frac{SN}{R(R-1)} mn)$	$O(mn)$	$O(1)$	$O(1)$
RDC	Polynomial-time in fixed dimension	$O(mn)$	$O(1)$	$O(1)$
MRR	Polynomial-time in fixed dimension	$O(mn)$	$O(1)$	$O(1)$

Table IV: Parameter settings.

Parameter	Meaning	Setting
N	# of servers	1000-10000
M	# of chunks of a data object	3 [53]
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

Table III shows the comparison of various replication schemes on various metrics. RDC and MRR optimize a NLIP problem. The objective functions of the optimization problem in RDC and MRR are convex min, and the constraints of the optimization problem are convex. Thus, the computational complexity of RDC and MRR is in polynomial-time in fixed dimension [50].

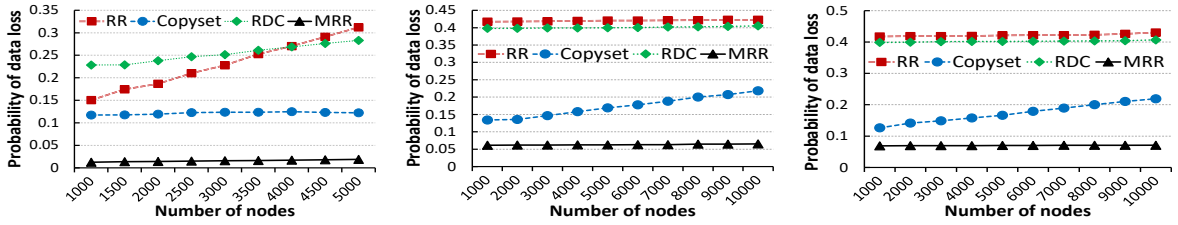
VI. Performance Evaluation

We conducted the numerical analysis based on the parameters in [18] (Table II) derived from the system statistics from Facebook, HDFS and RAMCloud [1], [2], [15], [18], [19], [51], 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 [52] provided by Sandia National Laboratories that records 4-hour read/write log in a parallel file system.

A. Numerical Analysis Results

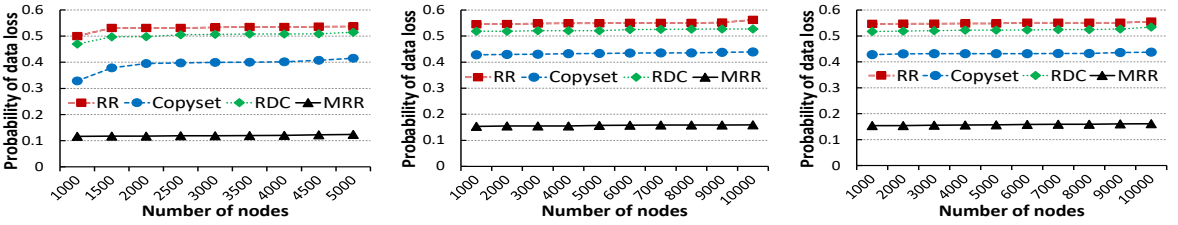
We compared MRR with other three replication schemes: Random Replication (RR) [18], Replication Degree Customization (RDC) [13] and Copyset Replication [18] (Copyset). RR places the primary replica on a random node (say node i) in the entire system, and places the secondary replicas on (R-1) nodes around 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

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



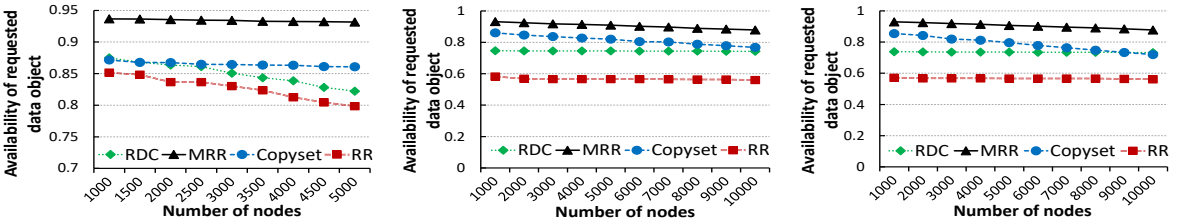
(a) Facebook (b) HDFS (c) RAMCloud

Figure 5: Probability of data loss vs. number of nodes ($R = 3$ for RR and Copyset).



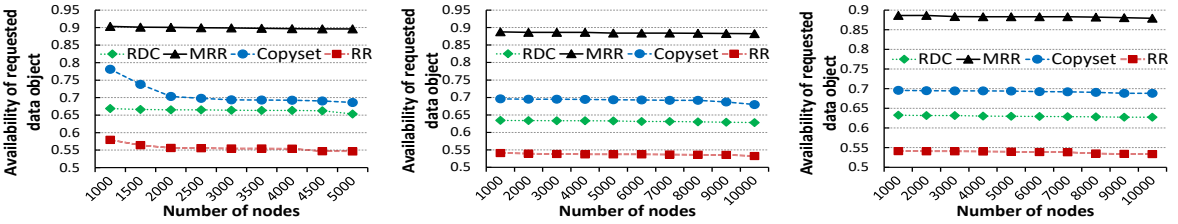
(a) Facebook (b) HDFS (c) RAMCloud

Figure 6: Probability of data loss vs. number of nodes ($R = 2$ for RR and Copyset).



(a) Facebook (b) HDFS (c) RAMCloud

Figure 7: Availability of requested data objects vs. number of nodes ($R = 3$ for RR and Copyset).



(a) Facebook (b) HDFS (c) RAMCloud

Figure 8: Availability of requested data objects vs. number of nodes ($R = 2$ for RR and Copyset).

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 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 [18]. Since this rate is the maximum percentage of concurrent failure nodes in real clouds (i.e., worst case) [15]–[17], 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 [13]. 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 IV 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 5(a) - 5(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. MRR achieves up to 36% lower data loss probability compared to the other three methods. 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,

[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 [18].

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.

To further verify MRR's performance in reducing the probability of data loss, we changed the value of R from 3 to 2, and kept the other parameters' settings and formulas for calculating four methods' data loss probabilities the same as Figure 5. Figure 6(a), 6(b) and 6(c) show the relationship between the probability of data loss and the number of nodes in Facebook, HDFS and RAMCloud environments with $R = 2$, respectively. Similar to Figure 5, we also see that the probability of data loss follows MRR<Copyset<RDC<RR due to the same reasons. MRR achieves up to 42% lower data loss probability compared to the other three methods. Both the result in Figure 5 and the result in Figure 6 confirm that MRR generates the lowest probability of data loss. Comparing Figure 6 to Figure 5, we find that the probability of data loss decreases as the number of replicas R increases. This is because the more the replicas for a chunk, the lower the probability that all the machines storing the chunk fail simultaneously, and thus the lower the probability of the chunk being lost.

We then calculate the availability of requested data object by $1-\bar{P}_r$, and $\bar{P}_r$ is calculated by Formula (9). Figure 7(a) - 7(c) show the result of the availability of requested data objects. We see the result generally follows MRR>Copyset>RDC>RR in all figures. MRR achieves up to 37% higher availability of requested data objects compared to the other three methods. The availability of requested data objects in Copyset is lower than that in MRR because MRR considers 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 also varied R from 3 to 2 to better verify the availability of MRR. Figure 8(a), 8(b) and 8(c) show the relationship between the availability of requested data objects and the number of nodes in the Facebook, HDFS and RAMCloud environments with $R = 2$, respectively. Similar to Figure 7, we also see that the availability of requested data objects follows MRR>Copyset>RDC>RR, and MRR achieves up to 35% higher availability of requested data objects compared to the other three methods, which is consistent with the result in Figure 7. Comparing Figure 8 with Figure 7, we find that the availability of requested data objects increases as the number of replicas R increases. This is because the more replicas for a data object, the lower the probability of the data object being lost, thus the higher the availability of the requested data object.

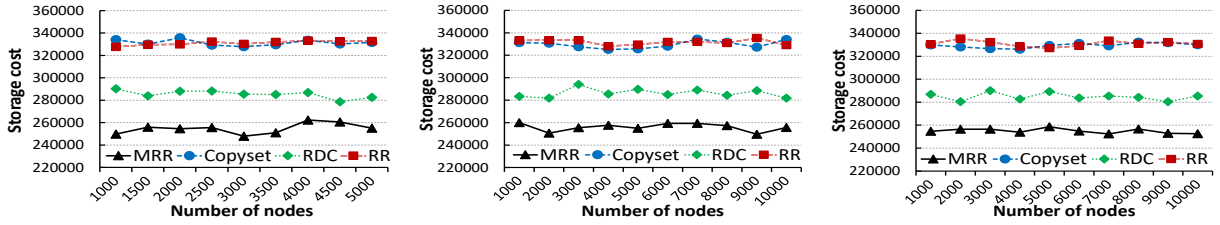
We then used Formula (15) to calculate the storage cost based on the sizes, replication degrees, and storage medium unit costs of data objects for MRR. For the other three methods, we randomly choose storage media for data objects and do not minimize the storage cost with space capacity constraint (Inequ. (17)) for RDC. Figure 9(a) - 9(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. These results indicate the lower storage cost of MRR by considering both the data popularity and storage medium cost in replication.

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 10(a) - 10(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. These results indicate MRR generates much lower consistency maintenance cost than the other methods.

B. Real-world Experimental Results

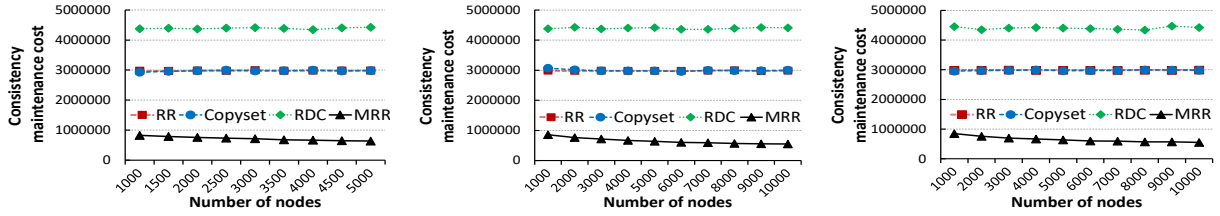
We conducted experiments on the real-world Amazon S3. We simulated the geo-distributed storage datacenters using three regions of Amazon S3 in the U.S. In each region, we created the same number of buckets, each of which simulates a data server. The number of buckets was varied from 5 to 25 with step 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 total size of the data objects is 24.4GB. 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 IV 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

[8]Since it is hard to generate permanent failures on Amazon S3 and the network latency is low based on [54], we assume request failure 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.



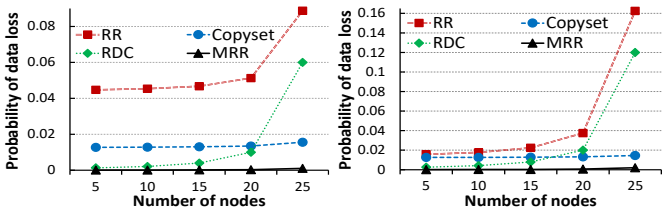
(a) Facebook (b) HDFS (c) RAMCloud

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



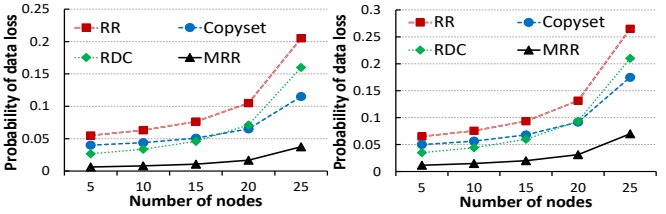
(a) Facebook (b) HDFS (c) RAMCloud

Figure 10: Consistency maintenance cost vs. number of nodes ($R = 3$ for RR and Copyset).



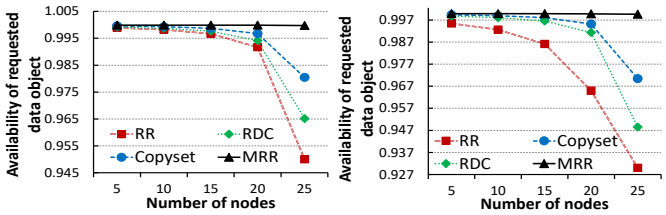
(a) Scatter width=2 (b) Scatter width=4

Figure 11: Probability of data loss vs. number of nodes on Amazon S3 ($R = 3$ for RR and Copyset).



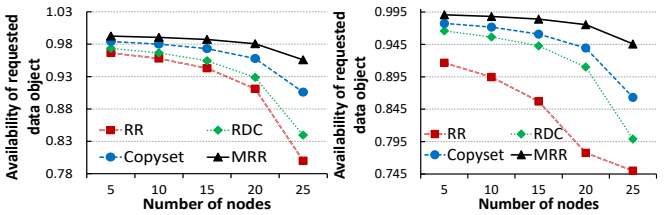
(a) Scatter width=2 (b) Scatter width=4

Figure 12: Probability of data loss vs. number of nodes on Amazon S3 ($R = 2$ for RR and Copyset).



(a) Scatter width=2 (b) Scatter width=4

Figure 13: Availability of requested data objects vs. number of nodes on Amazon S3 ($R = 3$ for RR and Copyset).



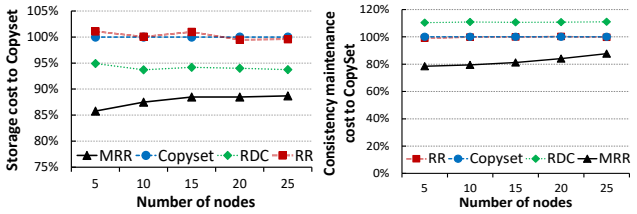
(a) Scatter width=2 (b) Scatter width=4

Figure 14: Availability of requested data objects vs. number of nodes on Amazon S3 ($R = 2$ for RR and Copyset).

system [8], [9], [55]. We used the actual price of the data access of Amazon S3 to calculate the storage cost.

Figure 11(a) and 11(b) show the result of probability of data loss on Amazon S3 when the scatter width (S) equals 2 and 4, respectively. We see the result approximately follows MRR<RDC<Copyset<RR. MRR achieves up to 16% lower data loss probability compared to the other three methods. 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. We then decreased the value of R to 2 in Figure 12. Figure 12 mirrors Figure 11 due to the same reasons. Comparing Figure 12 with Figure 11, we find that the probability of data loss decreases as R increases because the larger the number of replicas for a chunk, the lower the probability that all the machines storing the chunk fail concurrently, and thus the lower the probability that the chunk is lost.

Figure 13(a) and 13(b) show result of the availability of requested data objects on Amazon S3 when the scatter width equals 2 and 4, respectively. We see the result follows MRR>Copyset>RDC>RR, and MRR achieves up to 7% higher availability of requested data objects compared to the other three methods, which is consistent with the result in Fig-



(a) Storage cost (b) Consistency maintenance cost

Figure 15: Storage/consistency maintenance cost vs. number of nodes on Amazon S3 ($R = 3$ for RR and Copyset).

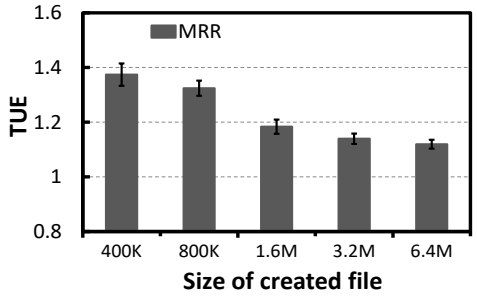


Figure 16: Relationship between TUE and size of the created file.

ure 7 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 data loss probability and reduces data availability. We also decreased R to 2 in Figure 14. Figure 14 mirrors Figure 13 due to the same reasons. Comparing Figure 14 and 13, we find the availability of requested data objects increases as R increases due to the similar reasons explained in the comparison of Figure 12 and 11.

We then regard Copyset as a baseline and calculate the ratio of the storage cost and consistency maintenance cost of each of the other methods over that of Copyset. Figure 15(a) and 15(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 MRR achieves up to 15.3% lower storage cost ratio compared to the other three methods. The consistency maintenance cost ratio follows RDC>RR≈Copyset>MRR, and MRR achieves up to 32% lower consistency maintenance cost ratio compared to the other three methods. These are consistent with that in Figure 9 and 10 due to the same reasons.

To measure the Traffic Usage Efficiency (TUE) of network efficiency (data synchronization) of MRR, we borrow a novel metric named ***TUE*** ($TUE = \frac{Total\ data\ sync\ traffic}{Data\ update\ size}$) from the work [56]. Figure 16 shows the relationship between TUE and the size of created files in the servers. In Figure 16, we see that TUE decreases as the file size increases, and the drop rate decreases as the file size increases. As the file size increases, TUE eventually keeps stable.

C. Clustering and Mapping Performance

To verify the performance of the minimum cut tree-based clustering algorithm, we compared MRR with minimum cut tree clustering with the original method (MRR without minimum cut tree clustering) under various scenarios. In experiment, we used the 6-bit number to represent the distance between two data objects (as explained in Section III-B) and generated the distance randomly for each data object. We assume the request time of a data object is proportional to the distance between the front server and the data server which stores the data object. For the original method, we acquired m data objects of the application sequentially without clustering. Hence, the total request time can be formulated as:

$$T_{orig} = \sum_{i=1}^{m} t_i, \tag{26}$$

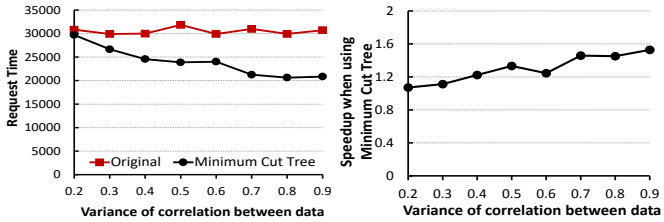
where t_i is the time for acquiring the ith data object, and is related to the size of the data object and the distance between the request server and the server that stores the data object.

In the minimum cut tree-based clustering algorithm, the data objects with high correlation would be grouped into the same cluster and be mapped onto the same data server. Therefore, for each data object request within one data cluster, the rest of the data objects in the same cluster would also be sent to the front server which decreases request time. In our experiment, the high correlation data is defined as $R_T(\chi_1, \chi_2) > 0.9$ because we assume the correlation is orthogonal, which means that $R_T(\chi_1, \chi_2) \in [0, 1]$. Then, we computed the request time for different clusters.

In order to simulate the correlation between data objects, we chose the Normal Distribution because most of the data objects have neither a high correlation nor extreme low correlation [57], [58]. Therefore, we believe it is reasonable to assume the correlation between data objects follows Normal Distribution. Moreover, we chose the Pareto Distribution for the same purpose [59], [60], and the results also do not change.

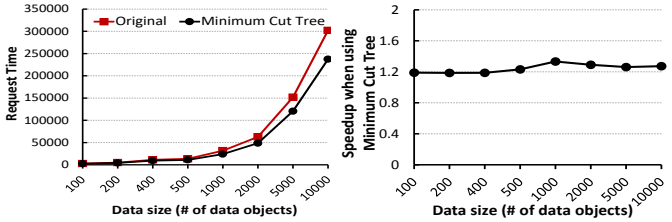
Figure 17(a) and 17(b) show the result of request time and speedup of data objects with the change of variance of correlation between data (standard deviation of distribution), respectively. The speedup is measured by dividing the request time in original method into that of MRR with minimum cut tree clustering. Figure 17(a) shows that the request time in original method remains stable as the variance of correlation increases. In contrast, the request time in MRR with minimum cut tree clustering decreases as the standard deviation increases. Figure 17(b) shows that the speedup in general increases the variance of correlation increases. This is because the loose distribution of data makes the proportion of high correlation data larger. Thus, MRR with minimum cut tree clustering outperforms the original method.

Figure 18(a) and 18(b) show the result of request time and speedup of data objects with the change of data size (number of data objects), respectively. The frequency of request is originally proportion to the data size. Thus, as shown in Figure 18(a), the request time increases as data size increases. Also, Figure 18(a) shows that the larger data size brings more time savings in MRR with minimum cut tree clustering. Figure 18(b) shows that the speedup is steady in general but slightly improved as the data size increases. Since the performance scales up with the data size, it can help further improve performance by applying the clustering algorithm to the large dataset in cloud platform. Although the performance of our algorithm could vary when it is applied to different applications with different data correlation distribution patterns,



(a) Request time (b) Speedup

Figure 17: Performance with and without data clustering when applying different standard deviation of correlation between data.



(a) Request time (b) Speedup

Figure 18: Performance with and without data clustering when applying different size of data in an application.

the results in Normal Distribution and Pareto Distribution pattern (which are two common and important distributions) show the efficiency of the proposed clustering method.

VII. 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 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 by considering the popularity of data. Moreover, we introduce the concept of the correlation between data objects to analyze the probability of two data objects being requested concurrently or sequentially to further reduce the replication cost. 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.

Acknowledgment

This research was supported in part by U.S. NSF grants OAC-1724845, ACI-1719397 and CNS-1733596, and Microsoft Research Faculty Fellowship 8300751. An early version of this work was presented in the Proceedings of HiPC 2016 [61]. We would like to thank Dr. Rajkumar Buyya, Dr. Adam Hoover and Dr. Svetlana Poznanović for their help on this work.

References

[1] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Proc. of MSST*, 2010, pp. 1–10.
[2] D. Ongaro, S. Rumble, R. Stutsman, J. Ousterhout, and M. Rosenblum, "Fast crash recovery in ramcloud," in *Proc. of SOSP*, 2011, pp. 29–41.
[3] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *Proc. of SOSP*, 2003, pp. 29–43.
[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] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: Measurement, analysis, and implications," in *Proc. of SIGCOMM*, Toronto, August 2011.
[7] W. Guo, K. Chen, Y. Wu, and W. Zheng, "Bidding for highly available services with low price in spot instance market," in *Proc. of HPDC*, 2015.
[8] D. Poola, S. Garg, R. Buyya, Y. Yang, and K. Ramamohanarao, "Robust scheduling of scientific workflows with deadline and budget constraints in clouds," in *Proc. of AINA*, 2014.
[9] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "Dcell: a scalable and fault-tolerant network structure for data centers," in *Proc. of SIGCOMM*, Seattle, 2008.
[10] S. Samundiswary and N. M. Dongre, "Object storage architecture in cloud for unstructured data," in *Proc. of ICISC*, Coimbatore, 2017.
[11] K. Eaton, "How one second could cost amazon $1.6 billion in sales," 2012. [Online]. Available: https://www.fastcompany.com/1825005/how-one-second-could-cost-amazon-16-billion-sales
[12] S. Nath, H. Yu, P. Gibbons, and S. Seshan, "Subtleties in tolerating correlated failures in wide-area storage systems," in *Proc. of NSDI*, 2006.
[13] M. Zhong, K. Shen, and J. Seiferas, "Replication degree customization for high availability," in *Proc. of EuroSys*, Glasgow, 2008.
[14] A. Haeberlen, A. Mislove, and P. Druschel, "Glacier: Highly durable, decentralized storage despite massive correlated failures," in *Proc. of NSDI*, 2005.
[15] R. Chansler, "Data availability and durability with the hadoop distributed file system," *The USENIX Magazine*, 2012.
[16] J. Dean, "Evolution and future directions of large-scale storage and computation systems at google," in *Proc. of SoCC*, 2010, p. 1.
[17] 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.
[18] 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 ATC*, 2013.
[19] 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.
[20] W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer, "Feasibility of a serverless distributed file system deployed on an existing set of desktop pcs," in *Proc. of SIGMETRICS*, 2000.
[21] 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.
[22] L. Yang, Y. Wang, D. Dunne, M. Sobolev, M. Naaman, and D. Estrin, "More than just words: Modeling non-textual characteristics of podcasts," in *Proc. of WSDM*, 2019.
[23] E. Keller, J. Szefer, J. Rexford, and R. B. Lee, "Nohype: Virtualized cloud infrastructure without the virtualization," in *Proc. of ISCA*, 2010.
[24] N. Bonvin, T. Papaioannou, and K. Aberer, "A self-organized, fault-tolerant and scalable replication scheme for cloud storage," in *Proc. of SoCC*, 2010.
[25] H. Yu and A. Vahdat, "The costs and limits of availability for replicated services," in *Proc. of SOSP*, 2001, pp. 29–42.
[26] C. Tang and S. Dwarkadas, "Hybrid global-local indexing for efficient peer-to-peer information retrieval," in *Proc. of NSDI*, 2004.
[27] 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*, 2013, pp. 1276–1284.
[28] H. Abu-Libdeh, R. Renesse, and Y. Vigfusson, "Leveraging sharding in the design of scalable replication protocols," in *Proc. of SoCC*, 2013.
[29] S. Y. Ko, I. Hoque, B. Cho, and I. Gupta, "Making cloud intermediate data fault-tolerant," in *Proc. of SoCC*, Indianapolis, 2010.
[30] E. Zhai, R. Chen, D. Wolinsky, and B. Ford, "Heading off correlated failures through independence-as-a-service," in *Proc. of OSDI*, 2014.

[31] E. Zhai, R. Piskac, R. Gu, X. Lao, and X. Wang, "An auditing language for preventing correlated failures in the cloud," in *Proc. of OOPSLA*, Vancouver, 2017.

[32] H. Yu, P. Gibbons, and S. Nath, "Availability of multi-object operations," in *Proc. of NSDI*, 2006, pp. 211–224.

[33] Q. Zhang, S. Li, Z. Li, Y. Xing, Z. Yang, and Y. Dai, "Charm: A cost-efficient multi-cloud data hosting scheme with high availability," *IEEE Trans. Cloud Comput.*, vol. 3, no. 3, pp. 372–386, 2015.

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

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

[36] H. Shen, Z. Li, Y. Lin, and J. Li, "SocialTube: P2P-assisted Video Sharing in Online Social Networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 9, pp. 2428–2440, 2014.

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

[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] Y. Xiang, T. Lan, V. Aggarwal, and Y. Chen, "Joint latency and cost optimization forerasure-coded data center storage," *IEEE/ACM Trans. Netw.*, vol. 24, no. 4, pp. 2443–2457, 2016.

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

[42] (IPOPT. https://projects.coin-or.org/Ipopt [accessed in Apr. 2019])

[43] A. Wächter and L. Biegler, "On the implementation of an interior-point filter linesearch algorithm for large-scale nonlinear programming," *Math. Program*, vol. 106, no. 1, 2006.

[44] T. Kraska, G. Pang, M. Franklin, S. Madden, and A. Fekete, "MDCC: Multi-data center consistency," in *Proc. of EuroSys*, Prague, 2013.

[45] J. Meza, T. Xu, K. Veeraraghavan, and O. Mutlu, "A large scale study of data center network reliability," in *Proc. of IMC*, Boston, 2018.

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

[47] Y. Hua, H. Jiang, and D. Feng, "Fast: Near real-time searchable data analytics for the cloud," in *Proc. of SC*, 2014.

[48] Y. Hu, Z. Jin, H. Ren, D. Cai, and X. He, "Iterative multi-view hashing for cross media indexing," in *Proc. of MM*, Orlando, 2014.

[49] R. Möhring, A. Schulz, F. Stork, and M. Uetz, "Solving project scheduling problems by minimum cut computations," *Manage. Sci.*, vol. 49, no. 3, pp. 330–350, 2003.

[50] J. L. R. W. R. Hemmecke, M. Köppe, "Nonlinear integer programming," *50 Years of Integer Programming 1958-2008*, pp. 561–618, 2010.

[51] (Intelligent block placement policy to decrease probability of data loss. https://issues.apache.org/jira/browse/HDFS-1094 [accessed in Sept. 2019])

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

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

[54] "S3 faqs," https://aws.amazon.com/s3/faqs/ [accessed in Sept. 2019].

[55] Q. Wei, B. Veeravalli, B. Gong, L. Zeng, and D. Feng, "Cdrm: A cost-effective dynamic replication management scheme for cloud storage cluster," in *Proc. of CLUSTER*, 2010.

[56] Z. Li, C. Jin, T. Xu, C. Wilson, Y. Liu, L. Cheng, Y. Liu, Y. Dai, and Z. Zhang, "Towards network-level efficiency for cloud storage services," in *Proc. of IMC*, 2014.

[57] R. Vilaça, R. Oliveira, and J. Pereira, "A correlation-aware data placement strategy for key-value stores," in *Proc. of DAIS*, 2011.

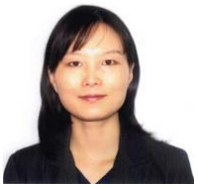
[58] Z. Ye, S. Li, and X. Zhou, "GCplace: Geo-cloud based correlation aware data replica placement," in *Proc. of SAC*, 2013.

[59] B. Arnold, *Pareto Distributions Second Edition*, ser. Chapman & Hall/CRC Monographs on Statistics & Applied Probability. Taylor & Francis, 2015.

[60] T. Armstrong, V. Ponnekanti, D. Borthakur, and M. Callaghan, "Linkbench: a database benchmark based on the facebook social graph," in *Proc. of SIGMOD*, 2013.

[61] 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.

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

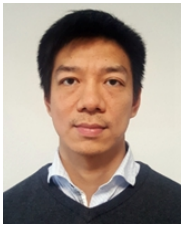


Haiying Shen received the BS degree in Computer Science and Engineering from Tongji University, China in 2000, and the MS and Ph.D. degrees in Computer Engineering from Wayne State University in 2004 and 2006, respectively. She is currently an Associate Professor in the Computer Science Department at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

Hongmei Chi Hongmei Chi received Ph.D. degree in Computer Science from Florida State University in 2004. She is currently a professor and the director of FAMU center for Cyber Security in the Department of Computer and Information Sciences at the Florida A&M University. Her research interests include big data, digital forensics, mobile security and data science. Her work in those areas has been published in top journals and conferences.

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

Yongyi Yang received her BS degree in Electrical Engineering from Nanjing University of Posts and Telecommunication, Nanjing, China, in 2016. She is currently a master student majoring in Computer Engineering in the University of Virginia, VA, United States. Her study interests include cloud computing and machine learning.



Long Cheng Long Cheng is an Assistant Professor in the School of Computing at Dublin City University, Ireland. He received the B.E. from Harbin Institute of Technology, China in 2007, M.Sc from University of Duisburg-Essen, Germany in 2010 and Ph.D from National University of Ireland Maynooth in 2014. He was a Marie Curie Fellow at University College Dublin. His research focuses on high performance data analytics, distributed systems and process mining. He is a member of the IEEE.

Wingyan Chung Wingyan Chung received his PhD in Management Information Systems from University of Arizona and his MS and BBA from Chinese University of Hong Kong. He is currently a Professor in the Institute for Simulation and Training at University of Central Florida. His research interests include social media analytics, business intelligence, network science, and simulation. He has over 90 refereed publications. He has received over $7 million from NSF, DARPA, DHS, among others. He is a Fulbright US Scholar and is ranked among Top 20 Researchers in Business Intelligence and Analytics.