CHARM: A Cost-Efficient Multi-Cloud Data Hosting Scheme with High Availability

Quanlu Zhang, Shenglong Li, Zhenhua Li, *Member, IEEE*, Yuanjian Xing, Zhi Yang, and Yafei Dai, *Member, IEEE*

Abstract—Nowadays, more and more enterprises and organizations are hosting their data into the cloud, in order to reduce the IT maintenance cost and enhance the data reliability. However, facing the numerous cloud vendors as well as their heterogenous pricing policies, customers may well be perplexed with which cloud(s) are suitable for storing their data and what hosting strategy is cheaper.The general status quo is that customers usually put their data into a single cloud (which is subject to the vendor lock-in risk) and then simply trust to luck. Based on comprehensive analysis of various state-of-the-art cloud vendors, this paper proposes a novel data hosting scheme (named CHARM) which integrates two key functions desired. The first is selecting several suitable clouds and an appropriate redundancy strategy to store data with minimized monetary cost and guaranteed availability. The second is triggering a transition process to re-distribute data according to the variations of data access pattern and pricing of clouds. We evaluate the performance of CHARM using both trace-driven simulations and prototype experiments. The results show that compared with the major existing schemes, CHARM not only saves around 20 percent of monetary cost but also exhibits sound adaptability to data and price adjustments.

Index Terms—Multi-cloud, data hosting, cloud storage

1 Introduction

Recent years have witnessed a "gold rush" of *online data hosting services* (or says *cloud storage services*) such as Amazon S3, Windows Azure, Google Cloud Storage, Aliyun OSS [1], and so forth. These services provide customers with reliable, scalable, and low-cost data hosting functionality. More and more enterprises and organizations are hosting all or part of their data into the cloud, in order to reduce the IT maintenance cost (including the hardware, software, and operational cost) and enhance the data reliability [2], [3], [4]. For example, the United States Library of Congress had moved its digitized content to the cloud, followed by the New York Public Library and Biodiversity Heritage Library [5]. Now they only have to pay for exactly how much they have used.

Heterogenous clouds. Existing clouds exhibit great heterogeneities in terms of both working performances and pricing policies. Different cloud vendors build their respective infrastructures and keep upgrading them with newly emerging gears. They also design different system architectures and apply various techniques to make their services competitive. Such system diversity leads to observable performance variations across cloud vendors [6].

Moreover, pricing policies of existing storage services provided by different cloud vendors are distinct in both pricing levels and charging items. For instance, Rackspace does not charge for Web operations (typically via a series of REST ful APIs), Google Cloud Storage charges more for bandwidth consumption, while Amazon S3 charges more for storage space (refer to Section 2.1).

Vendor lock-in risk. Facing numerous cloud vendors as well as their heterogenous performances/policies, customers may be perplexed with which cloud(s) are suitable for storing their data and what hosting strategy is cheaper.The general status quo is that customers usually put their data into a single cloud and then simply trust to luck. This is subject to the so-called "*vendor lock-in risk*", because customers would be confronted with a dilemma if they want to switch to other cloud venders.

The vendor lock-in risk first lies in that data migration inevitably generates considerable expense. For example, moving 100 TB of data from Amazon S3 (California datacenter) to Aliyun OSS (Beijing datacenter) would consume as much as 12,300 (US) dollars.

Besides, the vendor lock-in risk makes customers suffer from price adjustments of cloud vendors which are not uncommon. For example, the fluctuation of electricity bills in a region will affect the prices of cloud services in this region. We notice that giant cloud vendors like Windows Azure and Google Cloud Storage have been adjusting their pricing terms [7], [8].

Unexpected bankruptcy of cloud vendors further aggravates the situation. Nirvanix, which has thousands of customers including top 500 companies, suddenly shut down its cloud storage service in Sep. 2013 [9]. Ubuntu One, also a famous player in the market of cloud storage service, escaped in Apr. 2014 [10]. So clearly, it is unwise for an enterprise or an organization to host all data in a single cloud—"*your best bet is probably not to put all your eggs in one basket*." [11]

- *Q. Zhang, S. Li, Z. Yang, and Y. Dai are with Peking University, Beijing, China. E-mail: {zql, lishenglong, yangzhi, dyf}@net.pku.edu.cn.*
- *Z. Li is with Tsinghua University, Beijing, China. E-mail: lizhenhua1983@tsinghua.edu.cn.*
- *Y. Xing is with the Nanjing Research Institute of Electronics Technology, Nanjing, China. E-mail: xyj@net.pku.edu.cn.*

Manuscript received 12 June 2014; revised 23 Dec. 2014; accepted 5 Mar. 2015. Date of publication 29 Mar. 2015; date of current version 4 Sept. 2015. Recommended for acceptance by B. He and B. Veeravalli.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TCC.2015.2417534

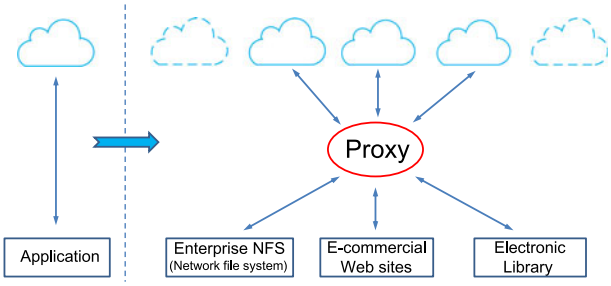


Fig. 1. Basic principle of multi-cloud data hosting.

Finally, uncontrolled data availability is (in a sense) another type of vendor lock-in risk. Though the service quality is formally guaranteed by service level agreements (SLA), failures and outages do occur. Almost all the major cloud vendors experienced service outages in recent years [12], [13], [14]. Some outages even lasted for several hours.

Multi-cloud data hosting. Recently, *multi-cloud data hosting* has received wide attention from researchers, customers, and startups. The basic principle of multi-cloud (data hosting) is to distribute data across multiple clouds to gain enhanced redundancy and prevent the vendor lock-in risk, as shown in Fig. 1. The "proxy" component plays a key role by redirecting requests from client applications and coordinating data distribution among multiple clouds.

The potential prevalence of multi-cloud is illustrated in three folds. First, there have been a few researches conducted on multi-cloud. DepSky guarantees data availability and security based on multiple clouds, thus allowing critical data (e.g., medical and financial data) to be trustingly stored [15]. RACS deploys erasure coding among different clouds in order to prevent vender lock-in risk and reduce monetary cost [16]. Second, new types of cloud vendors (e.g., DuraCloud [17] and Cloud Foundry [18]) have emerged and rapidly grown up to provide real services based on multiple clouds. Third, new development tools like Apache libcloud [19] provide a unified interface above different clouds, which facilitates migrating services among clouds.

Nevertheless, as for multi-cloud people still encounter the two critical problems: (1) *How to choose appropriate clouds to minimize monetary cost in the presence of heterogenous pricing policies?* (2) *How to meet the different availability requirements of different services?* As to monetary cost, it mainly depends on the data-level usage, particularly storage capacity consumption and network bandwidth consumption. As to availability requirement, the major concern lies in which redundancy mechanism (i.e., replication or erasure coding) is more economical based on specific data access patterns. In other words, here the fundamental challenge is: *How to combine the two mechanisms elegantly so as to greatly reduce monetary cost and meanwhile guarantee required availability?*

The proposed CHARM scheme. In this paper, we propose a novel **c**ost-efficient data **h**osting scheme with high **a**vailability in hete**r**ogenous **m**ulti-cloud, named "CHARM". It intelligently puts data into multiple clouds with minimized monetary cost and guaranteed availability.

Specifically, we combine the two widely used redundancy mechanisms, i.e., replication and erasure coding, into a uniform model to meet the required availability in the presence of different data access patterns. Next, we design an efficient heuristic-based algorithm to choose proper data storage modes (involving both clouds and redundancy mechanisms). Moreover, we implement the necessary procedure for storage mode transition (for efficiently re-distributing data) by monitoring the variations of data access patterns and pricing policies.

We evaluate the performance of CHARM using both trace-driven simulations and prototype experiments. The traces are collected from two online storage systems: AmazingStore [20] and Corsair [21], both of which possess hundreds of thousands of users. In the prototype experiments, we replay samples from the two traces for a whole month on top of four mainstream commercial clouds: Amazon S3, Windows Azure, Google Cloud Storage, and Aliyun OSS. Evaluation results show that compared with the major existing schemes (i.e., RepRa [22], RepGr [23], EraRa [16], and EraGr [24] which will be elaborated in Section 7.2), CHARM not only saves around 20 percent (more in detail, 7-44 percent) of monetary cost but also exhibits sound adaptability to data and price adjustments.

Summary of contribution. At last, our contributions in this paper can be briefly summarized as follows:

1) We propose and implement CHARM, a novel, efficient, and heuristic-based data hosting scheme for heterogenous multi-cloud environments. CHARM accommodates different pricing strategies, availability requirements, and data access patterns. It selects suitable clouds and an appropriate redundancy strategy to store data with minimized monetary cost and guaranteed availability.
2) We design and implement a flexible transition scheme for CHARM. It keeps monitoring the variations of pricing policies and data access patterns, and adaptively triggers the transition process between different data storage modes. It also starts a data migration process among different clouds if necessary.
3) We evaluate the performance of CHARM using two typical real-world traces and prototype experiments. Both trace-driven simulation and experiment results confirm the efficacy of CHARM.

Roadmap. The remainder of this paper is organized as follows. First, we briefly introduce the pricing models of mainstream cloud vendors and the basic knowledge of erasure coding in Section 2. Then, we demonstrate the new opportunity of multi-cloud by combining replication and erasure coding in Section 3. In Sections 4 and 5, we present the architecture and two key components of CHARM. Section 6 discusses the practical issues of CHARM. After that, we evaluate the performance of CHARM in Section 7 and review related work in Section 8. Finally, we conclude the paper in Section 9.

2 Background

2.1 Pricing Models of Mainstream Clouds

In order to understand the pricing models of mainstream cloud vendors, we select to study five most popular cloud storage services across the world: Amazon S3, Windows

TABLE 1
Prices of Storage (in $/GB/Month) and Out-Going Bandwidth (in $/GB)

Clouds	Amazon S3			Windows Azure			Google Cloud Storage		Rackspace	Aliyun OSS
	Tokyo	Singapore	America	US East	US RA-GRS	Asia	America	Asia Pacific	All Regions	China
Storage	0.033	0.03	0.03	0.0243	0.0616	0.024	0.026	0.026	0.105	0.028
Out-going Bandwidth	0.201	0.19	0.12	0.12	0.12	0.19	0.12	0.21	0.2	0.116

Azure, Google Cloud Storage, Rackspace, and Aliyun OSS (deployed in China). Their latest pricing models (in 2014) are presented in Table 1 (Storage and bandwidth pricing) and Table 2 (Operation pricing).

Basically for these clouds, customers are charged in terms of storage, out-going (i.e., from cloud to client) bandwidth[1], and operations (such as PUT, GET, and LIST). However, each vendor's pricing model has some difference from the others. For instance, in Asia Amazon S3 has lower bandwidth price and higher storage price than Google Cloud Storage. Aliyun OSS provides the lowest bandwidth price, but its storage price is still higher than Google Cloud Storage. Besides, prices of operations are also different across different clouds, as shown in Table 2.

2.2 Erasure Coding

Erasure coding has been widely applied in storage systems in order to provide high availability and reliability while introducing low storage overhead [25]. As we all know, the storage mode of "three replicas" is putting replicas into three different storage nodes. Then the data is lost only when the three nodes all crash. However, it occupies 2x more storage space. Erasure coding is proposed to reduce storage consumption greatly while guaranteeing the same or higher level of data reliability.

A representative erasure-coding scheme is the so-called "Reed-Solomon code", which is a type of *Maximum Distance Separable* erasure coding. Considering a storage system with M nodes, we divide data into blocks of equal size, and each block is further divided into m equal-sized data chunks. After that, we encode the m chunks into $n-m$ parity chunks and put the total n chunks into different nodes ($n \leq M$). We use RS(m, n) to denote this coding scheme and we call the n chunks a "segment".

More in detail, each parity chunk is generated by a linear combination of the m data chunks. Let $\{\gamma_{i,j}\}_{1\leq i\leq n-m, 1\leq j\leq m}$ be the coefficients of the linear combinations and let $\{D_i\}_{1\leq i\leq m}$ and $\{P_i\}_{1\leq i\leq n-m}$ denote the data chunks and parity chunks, respectively. Then, $P_i = \sum_{j=1}^{m} \gamma_{i,j} D_j$, where all the arithmetic operations are performed in the Galois Field.

Through the abovementioned efforts, the Reed-Solomon coding scheme can tolerate up to $n-m$ simultaneous chunk failures. In other words, we can restore data using any m chunks in the segment. If $m = 1$, RS(1, $n-1$) is just the traditional replication scheme, where $\gamma_{i,j} = 1$, $\forall i, j$, and we have n replicas for each data block.

In order to encode data, we have to implement *multiplication* operations in the Galois Field, which brings much computational complexity to the storage system, making computation a bottleneck. However, the situation can be significantly improved by using Intel SIMD instructions [26], perpetuating a trend of worrying more about I/O than CPU performance (see Section 6 for more details).

1. In-going (i.e., from client to cloud) bandwidth is not charged.

3 A New Opportunity in Multi-Cloud Storage

In this section, from a quantitative perspective, we demonstrate that there is still plenty of space for optimizing the multi-cloud data hosting by combining the two widely used redundancy mechanisms, i.e., replication and erasure coding.

3.1 Combining Replication and Erasure Coding

In existing industrial data hosting systems, data availability (and reliability) are usually guaranteed by replication or erasure coding. In the multi-cloud scenario, we also use them to meet different availability requirements, but the implementation is different. For replication, replicas are put into several clouds, and a read access is only served (unless this cloud is unavailable then) by the "cheapest" cloud that charges minimal for out-going bandwidth and GET operation.

For erasure coding, data is encoded into n blocks including m data blocks and $n-m$ coding blocks, and these blocks are put into n different clouds. In this case, though data availability can be guaranteed with lower storage space (compared with replication), a read access has to be served by multiple clouds that store the corresponding data blocks. Consequently, erasure coding cannot make full use of the cheapest cloud as what replication does. Still worse, this shortcoming will be amplified in the multi-cloud scenario where bandwidth is generally (much) more expensive than storage space.

3.2 Comparison of Data Hosting Modes

The traditional view of replication and erasure coding [27], [28] does not hold in the multi-cloud scenario. For example, the biggest preponderance of erasure coding lies in much less storage space for guaranteed high availability.

TABLE 2
Prices of Operations (in 10^{-5} Dollars/Operation)

Clouds	PUT	GET	LIST	DELETE
Rackspace	free	free	free	free
Amazon S3	1	0.1	1	free
Windows Azure	0.01	0.01	0.01	free
Google Cloud Storage	1	0.1	1	free
Aliyun OSS	0.016	0.016	0.016	free

TABLE 3
Symbol Table

Symbol	Meaning
S	Size of a file
c_r	Number of reads of a file in a time length of t
P_{si}	Storage price of the ith cloud
P_{bi}	Outgoing bandwidth price of the ith cloud
P_{oi}	GET operation price of the ith cloud

However, this preponderance shrinks because of the clouds' pricing policies—bandwidth is (much) more expensive than storage space. For the same reason, in the multi-cloud scenario replication regains its competitiveness, though it is traditionally regarded as inferior to erasure coding in terms of storage saving. Therefore, it is difficult now to determine which mechanism is better in the presence of complex workload patterns and various pricing policies. Below we compare the two mechanisms quantitatively to shed light on this problem.

In a multi-cloud system, to tolerate simultaneous outages of k clouds, replication needs to place $k+1$ copies of an object into $k+1$ different clouds. In contrast, erasure coding divides an object into m blocks, and encodes them into $m+k$ blocks. Here we use $k=1$ for simplicity in the following analysis. For the sake of clarity, Table 3 lists the variables used in this section.

For the replication mechanism, the storage cost is $S\sum_{i=1}^{2} P_{s\alpha_i}$, where α_i is the index of a cloud. The bandwidth cost is $c_r S P_{b\beta}$, where β is the index of the cloud with minimal bandwidth price among the two clouds. Therefore, the total cost C_R can be expressed below:

$$C_R = S\sum_{i=1}^{2} P_{s\alpha_i} + c_r S P_{b\beta}.$$

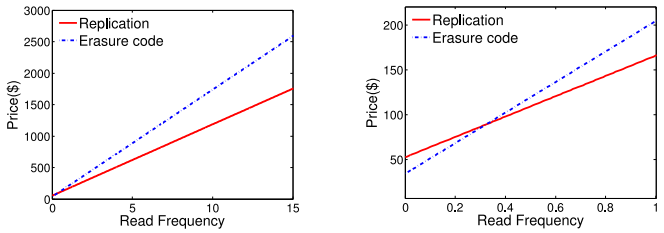
For the erasure coding mechanism, the storage cost is $\frac{S}{m}\sum_{i=1}^{m+1} P_{s\alpha_i}$, where α_i also represents the index of a cloud. The bandwidth cost is $\frac{c_r S}{m}\sum_{i=1}^{m} P_{b\beta_i}$, where β_i denotes the index of the cloud with the ith minimal outgoing bandwidth cost among the $m+1$ clouds. Thus, the total cost C_E can be expressed below:

$$C_E = \frac{S}{m}\sum_{i=1}^{m+1} P_{s\alpha_i} + \frac{c_r S}{m}\sum_{i=1}^{m} P_{b\beta_i}.$$

We use the data centers in Asia (including five data centers) to quantitatively compare the two redundancy mechanisms. The prices of storage and bandwidth are listed in Table 1. For replication, we pick Aliyun OSS which has the lowest bandwidth price and Windows Azure which has the lowest storage price. For erasure coding, we use all the five clouds. A more formal demonstration of cloud selection can be seen in Section 4.

Specifically, we artificially generate $100,000$ files, each of which is 10 MB in size. These files are stored for a month. Fig. 2 presents the monetary cost of replication and erasure coding with varying file read frequencies.

As shown in Fig. 2a, replication almost always outperforms erasure coding in the multi-cloud scenario. When the read count is 30, replication can save 32 percent monetary cost compared with erasure coding. The advantage of replication originates from the cloud with the lowest bandwidth price. In general, when the read frequency is high, the bigger the gap between the lowest bandwidth price and the average one is, the greater the superiority of replication is.



(a) Read frequency varies in 0-15 (b) Read frequency varies in 0-1

Fig. 2. Comparison between replication and erasure code when the read frequency varies.

On the contrary, as shown in Fig. 2b when the average read frequency falls below 0.32, erasure coding outperforms replication. This critical value seems small; however, we have to face the fact that as to cloud storage services, the majority of data only generates very small access workload, and data usually changes from "hot" to "cold". These facts degrade the effect of erasure coding to a similar level as that of replication. On the other hand, significant monetary saving can be achieved if we combine their advantages elegantly. This is why we propose the novel data hosting scheme CHARM which will be elaborated in the following sections.

4 Data Hosting Scheme

4.1 CHARM Overview

In this section, we elaborate a **c**ost-efficient data **h**osting model with high **a**vailability in hete**r**ogenous **m**ulti-cloud, named "CHARM". The architecture of CHARM is shown in Fig. 3. The whole model is located in the proxy in Fig. 1. There are four main components in CHARM: *Data Hosting*, *Storage Mode Switching (SMS)*, *Workload Statistic*, and *Predictor*.

Workload Statistic keeps collecting and tackling access logs to guide the placement of data. It also sends statistic information to *Predictor* which guides the action of *SMS*. *Data Hosting* stores data using replication or erasure coding,

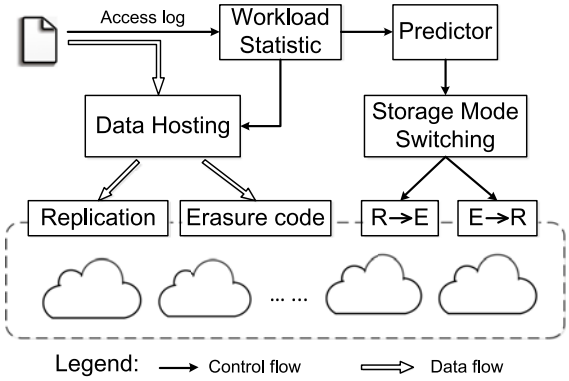


Fig. 3. The architecture of CHARM. "R" represents replication and "E" represents erasure coding.

according to the size and access frequency of the data. *SMS* decides whether the storage mode of certain data should be changed from replication to erasure coding or in reverse, according to the output of *Predictor*. The implementation of changing storage mode runs in the background, in order not to impact online service.

Predictor is used to predict the future access frequency of files. The time interval for prediction is one month, that is, we use the former months to predict access frequency of files in the next month. However, we do not put emphasis on the design of predictor, because there have been lots of good algorithms for prediction. Moreover, a very simple predictor, which uses the weighted moving average approach, works well in our data hosting model.

Data Hosting and *SMS* are two important modules in CHARM. *Data Hosting* decides storage mode and the clouds that the data should be stored in. This is a complex integer programming problem demonstrated in the following subsections. Then we illustrate how *SMS* works in detail in Section 5, that is, when and how many times should the transition be implemented.

4.2 Formal Definition of Data Hosting Model

We first formally define the mathematical model applied in *Data Hosting*. When talking about erasure coding, we usually mean $m > 1$ (not replication). However, replication is a special case of erasure coding (i.e., $m = 1$). So we combine the two storage mechanisms and define a unified model.

Assuming we have N clouds that meet performance requirements. We choose n cloud to store a file, the file should be encoded into n blocks of equal size ($n \leq N$), including m data blocks and $n - m$ coding blocks. If $m = 1$, the $n - m$ coding blocks are the same with the data block, i.e., replication. Then the n blocks are distributed into the n clouds. We call a (m, n) pair with its corresponding clouds a storage mode.

We first formally define the availability of a (m, n) pair. For the n clouds, which one stores data block, and which one stores coding block do not impact the availability. It is only impacted by the value of m.

We define $N' = \{i \in [1, N] | u_i = 1\}$, where $u_i \in \{0, 1\}$, $|N'| = n$. u_i is used to mark whether the ith cloud is chosen. N' is the set of the selected n clouds from the N available clouds for storing the file. It can tolerate simultaneous failures of any $n - m$ clouds. There are $\binom{|N'|}{k}$ cases of k simultaneously available clouds. We use $C_j^{|N'|,k}$ to denote the jth element in the set of the $\binom{|N'|}{k}$ cases. The total probability that there are k simultaneously available clouds can be expressed below:

$$Pr(N', k) = \sum_{j=1}^{\binom{|N'|}{k}} \left[\prod_{i \in C_j^{|N'|,k}} a_i \prod_{i \in N' \setminus C_j^{|N'|,k}} (1 - a_i) \right], \quad (1)$$

where a_i means the availability of the ith cloud.

Since the storage mode (m, n) can tolerate any $0 \sim (n - m)$ simultaneously failed clouds, its availability can be expressed as the sum of $Pr(N', k)$:

$$\sum_{k=m}^{|N'|} Pr(N', k) = \sum_{k=m}^{|N'|} \sum_{j=1}^{\binom{|N'|}{k}} \left[\prod_{i \in C_j^{|N'|,k}} a_i \prod_{i \in N' \setminus C_j^{|N'|,k}} (1 - a_i) \right]. \quad (2)$$

On the other hand, the monetary cost C_{sm} of a storage mode is composed of storage cost, bandwidth cost, and operation cost. The n clouds' storage costs compose the total storage cost, each cloud stores the data of size S/m. So the storage cost can be expressed below:

$$\sum_{i=1}^{N} \frac{S}{m} P_{si} u_i = \sum_{i \in N'} \frac{S}{m} P_{si}. \quad (3)$$

Since read access can be satisfied by only m clouds, we use the cheapest m clouds for read access. The m clouds are right used for storing data blocks while the left $n - m$ clouds for coding blocks. The normal read access does not need data decoding. Thus, the bandwidth and operation cost can be defined as

$$min_{j \in [1, \binom{|N'|}{m}]} \left\{ \sum_{i \in C_j^{|N'|,m}} \frac{S}{m} c_r P_{bi} + \sum_{i \in C_j^{|N'|,m}} c_r P_{oi} \right\}. \quad (4)$$

Actually, the expression of the operation cost above is not fair. For a large file (e.g., larger than 100 MB), we can split it into pieces, encode them, and put them into the clouds. When we request this file, we need to issue m GET requests. However, for a small file, such as several KB, we never split it into pieces again. Instead, we group many such files into a large block, then split and encode this block. In this case, getting a small file only uses one GET request. So we use $\frac{1}{m} \sum_{i \in C_j^{|N'|,m}} c_r P_{oi}$ as operation cost. Since operation cost has little impact for large files, this change does not affect the cost of requesting large files.

Therefore, if we use A to denote the required availability, the optimization problem can be formally defined as follows:

$$minC_{sm} = \sum_{i \in N'} \frac{S}{m} P_{si} + min_{j \in [1, \binom{|N'|}{m}]} \left\{ \sum_{i \in C_j^{|N'|,m}} \frac{S}{m} c_r P_{bi} + \frac{1}{m} \sum_{i \in C_j^{|N'|,m}} c_r P_{oi} \right\} \quad (5)$$

$$s.t. \quad \sum_{k=m}^{|N'|} \sum_{j=1}^{\binom{|N'|}{k}} \left[\prod_{i \in C_j^{|N'|,k}} a_i \prod_{i \in N' \setminus C_j^{|N'|,k}} (1 - a_i) \right] \geq A \quad (6)$$

$$N' = \{i \in [1, N] | u_i = 1\} \quad (7)$$

$$u_i \in \{0, 1\}, \forall i \in [1, N] \; and \; i \in Z^* \quad (8)$$

$$|N'| \geq m \geq 1, m \in Z^*. \quad (9)$$

This minimization problem is a complex integer programming problem. We first note that even the calculation

of Eq. (6) has the complexity of $O(2^{|N'|})$. More specifically, we have the following negative result:

Theorem 1. *The minimization problem described by Eqs. (5)-(9) is NP-hard.*

Proof. There are two steps. We first prove that the 0-1 integer linear programming is NP-complete by reducing it to the clique problem [29], then we prove our problem is much harder than 0-1 integer linear programming, is a NP-hard problem.

There is an undirected graph $G =< V, E >$ and non-negative integer J, where $J \leq |V|$ and $V = \{v_1, v_2, ..., v_n\}$. Finding a clique V' ($V' \in V$) of G and $|V'| \geq J$ can be reduced into the 0-1 integer programming as follows: Let $x_i = 1$ for each $v_i \in V'$. When $(v_i, v_j) \in E$, $x_i + x_j \leq 1$. $\sum_{i=1}^{n} x_i = |V'|$. So $|V'| \geq J$ becomes a decision problem whether $max \sum_{i=1}^{n} x_i \geq J$. The problem can be formally described as follows:

$$max \sum_{i=1}^{n} x_i$$

$$s.t. \quad x_i + x_j \leq 1, \quad \forall i, j \in \{1, 2, \ldots, n\}, i \neq j \; and \; (v_i, v_j) \in E$$

$$x_i \in \{0, 1\}, \forall i \in 1, 2, \ldots, n.$$

The reducing can be completed in polynomial execution time. If the 0-1 integer linear problem can be solve in polynomial time, the clique problem also has polynomial solution. So 0-1 integer linear programming is also NP-complete.

Our minimization problem is integer programming which is harder than 0-1 integer linear programming. Moreover, Eq (6) is a non-linear inequality which includes the case of linear inequality. Since the problem is harder than NP-complete, the minimization problem is NP-hard. □

In order to get the optimal solution, we have to traverse each combination of u_i, called a case. For each case we have to traverse m among feasible values. Then we check whether the case meets the availability requirement, and at last pick the case with the minimal cost. So, according to the calculation flow of this brute force approach, the complexity can be expressed below:

$$2^N \sum_{m=1}^{|N'|} \left(|N'| \sum_{k=m}^{|N'|} \binom{|N'|}{k} + \binom{|N'|}{m} \right). \tag{10}$$

The equation can be simplified as $O(2^N 2^{|N'|} |N'|^2)$. Obviously, this direct optimal approach will require inordinate amount of computation. In order to make the decision making practical and efficient, we demonstrate an effective heuristic solution below.

4.3 Heuristic Solution

The key idea of this heuristic algorithm can be described as follows:

We first assign each cloud a value δ_i which is calculated based on four factors (i.e., availability, storage, bandwidth, and operation prices) to indicate the preference of a cloud. We choose the most preferred n clouds, and then heuristically exchange the cloud in the preferred set with the cloud in the complementary set to search better solution. This is similar to the idea of Kernighan-Lin heuristic algorithm [30], which is applied to effectively partition graphs to minimize the sum of the costs on all edges cut.

The preference of a cloud is impacted by the four factors, and they have different weights. The availability is the higher the better, and the price is the lower the better. So we use $\delta_i = \alpha a_i + \frac{\beta}{P_i}$ as the preference of the ith cloud, where P_i is the synthetical price of storage, bandwidth, and operation. Intuitively, if a file has much read access, the cloud with lower bandwidth price is more preferred. If a file is very small, operation price occupies a big proportion. So we let $P_i = SP_{si} + c_r SP_{bi} + c_r P_{oi}$. Specifically, a_i and P_i are both normalized into $(0, 1)$.

Algorithm 1. Heuristic Algorithm of Data Placement

Input: file size S, read frequency c_r, n's upper limit ξ
Output: minimal cost C_{sm}, the set ψ of the selected clouds

$C_{sm} \leftarrow$ inf; $\psi \leftarrow \{\}$
$L_s \leftarrow$ sort clouds by normalized $\alpha a_i + \frac{\beta}{P_i}$ from high to slow
for n = 2 to ξ **do**
  $G_s \leftarrow$ the first n clouds of L_s
  $G_c \leftarrow L_s - G_s$
  for m = 1 to n **do**
    $A_{cur} \leftarrow$ calculate the availability of G_s
    if $A_{cur} \geq A$ **then**
      $C_{cur} \leftarrow$ calculate the minimal cost
      if $C_{cur} < C_{sm}$ **then**
        $C_{sm} \leftarrow C_{cur}$
        $\psi \leftarrow G_s$
      end
    else
      /*heuristically search better solution*/
      $G_s \leftarrow$ sort G_s by a_i from low to high
      $G_c \leftarrow$ sort G_c by P_i from low to high
      for i = 1 to n **do**
        flag $\leftarrow 0$
        for j = 1 to $N - n$ **do**
          if $a_{G_c[j]} > a_{G_s[i]}$ **then**
            swap $G_s[i]$ and $G_c[j]$
            flag $\leftarrow 1$
            break
          end
        end
        if *flag = 0* **then**
          break
        end
        $A_{cur} \leftarrow$ calculate the availability of G_s
        if $A_{cur} \geq A$ **then**
          $C_{cur} \leftarrow$ calculate the minimal cost
          if $C_{cur} < C_{sm}$ **then**
            $C_{sm} \leftarrow C_{cur}$
            $\psi \leftarrow G_s$
          end
          break
        end
      end
    end
  end
end
return C_{sm}, ψ

To find out optimal n and m, we first traverse n from 2 to ξ, where ξ is the upper limit of n and $\xi < N$. We do not set ξ to N for two reasons: reality and complexity. For reality, n tends to be small in practice, usually less than 10. It has much higher probability for large n to induce degraded performance. More specifically, if a cloud becomes unavailable, the proxy has to get corresponding data from other m clouds (m is usually close to n), which determines that n cannot be very large in order to achieve good performance. For complexity, calculating the availability of erasure coding (m, n) has very high complexity. We have to check the availability for every possible solution that is traversed. If we give an upper limit to n, the availability can be calculated in polynomial running time.

Then we traverse m from 1 to n for each n. The availability is calculated using Eq. 2. If the availability meets the required value and the monetary cost is lower, we update C_{sm} and ψ (i.e., the set of the selected clouds). If the availability does not meet the required value, we exchange the cloud in the current set G_s with the one in the complementary set G_c, using a greedy method: Firstly, G_s is sorted by a_i, and G_c is sorted by P_i. Then we try to exchange the cloud in G_s from the lowest a_i, one by one, with the cloud which has the lowest P_i in G_c but higher availability than that cloud in G_s, until the availability meets the required value. If the cost of the obtained G_s is lower, we update C_{sm} and ψ. The detailed process is shown in Algorithm 1.

4.4 Complexity

We analyze the complexity of this heuristic algorithm. The first two nested loops (line 3 and 6) are traversing n and m, which have the complexity of $O(\xi^2)$. The next two nested loops (line 18 and 20) are used to heuristically search for better solutions, the complexity is $O(\xi(N-\xi))$. Since the searching process stops once it finds a solution meeting the required availability, the complexity should be much lower than $O(\xi(N-\xi))$. The availability should be calculated in line 7 and 30 to check whether it meets the requirement. As Eq. (6) shows, the calculation has to iterate all the subsets, the sizes of which are equal to or larger than m, so the complexity is $O(2^n)$, where $n \leq \xi$.

For the calculation of the minimal cost (lines 9 and 32), we need to find out the cheapest subset of size m to serve read access, so sorting has to be implemented with the complexity of $O(\xi\log\xi)$. Then the sort operations in line 2, 16, and 17 are $O(N\log N)$, $O(\xi\log\xi)$, and $O((N-\xi)\log(N-\xi))$ respectively. Based on the analysis above, the worst case running time can be expressed as $O(N\log N + \xi^2(2^\xi + \xi\log\xi + \xi\log\xi + (N-\xi)\log(N-\xi) + \xi((N-\xi) + 2^\xi + \xi\log\xi)))$. Since ξ is a constant the computational complexity is actually $O(N\log N)$.

We then compare the complexity between CHARM and the brute force approach. Since n (i.e., $|N'|$) is bounded by an upper limit to make CHARM more efficient and practical, we also put this limit to the brute force approach. However, the upper limit would not give it the opportunity to be implemented in acceptable time. Since the upper limit is ξ, $O(2^N 2^{|N'|}|N'|^2)$ of the brute force approach can be expressed as $O(\sum_{i=1}^{\xi}\binom{N}{i}2^\xi\xi^2)$, which can be simplified as $O(N^\xi)$. Since ξ is relatively high, the brute force approach has much higher complexity which is a higher-degree polynomial with a big constant factor. Therefore, CHARM makes the data hosting decision much practical in a real system.

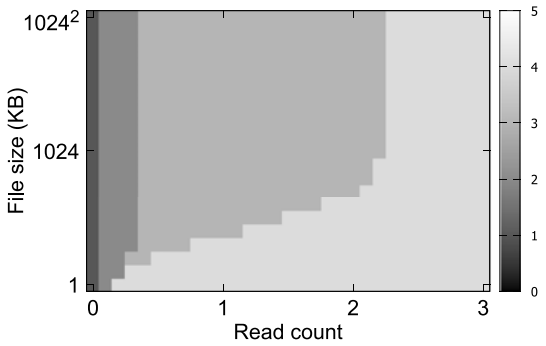


Fig. 4. The storage modes with different file size and read count. In the figure, there are four gray levels (1-4) which represent different storage modes. Gray level 1-4 are (7, 9), (7, 9), (6, 8), (1, 2) respectively. Gray levels 1 and 2 use different clouds though they have the same m and n.

We have demonstrated how to choose storage mode and clouds for a file with certain size and read frequency. Then we illustrate when and how many times should the storage mode be changed in the following section, since the access frequency of a file usually changes with time goes by.

5 Transition Scheme

5.1 Transition of Storage Modes

Intuitively, when a file changes from "hot" to "cold", we should change its storage mode. More specifically, when the read frequency of the file drops below or increases above a certain value, changing storage mode can save more money. The value is determined by the prices of clouds. Given the available clouds including their prices and availability, we can figure out the storage mode and the selected clouds with the input of file's size and read count, using Algorithm 1.

We calculate the storage modes for different file sizes and read counts, in order to get a storage mode table (see Fig. 4 in Section 7 for an example). The table has two dimensions: file size and read count. There is one corresponding storage mode for each pair of file size and read count, but the storage modes are the same for many different pairs. There are explicit boundaries between different storage modes in the table. However, it does not mean we should change the storage mode once a file's storage mode crosses the boundary, because the transition of storage mode also generates cost, which is definitely not negligible. Bandwidth is (much) more expensive than storage space for online storage services. The cost of one read access for a file can afford this file to be stored for around four months with no read access. Thus, we should be prudent to deal with storage mode transition. A good transition scheme can actually save large amount of money.

We first demonstrate the implementation of storage mode transition: the proxy gets the data from the clouds where the data is originally stored, and puts it into the newly selected clouds using new storage mode. The implementation consumes out-going bandwidth, in-going bandwidth, and several operations (i.e., GET, DELETE, and PUT). Since DELETE and in-going bandwidth are free, the transition cost T is composed of out-going bandwidth, GET, and PUT. Out-going bandwidth is more expensive than

storage, so we have to make sure that the cost of transition can be earned back by the new storage mode. That is, the following inequality has to be met:

$$M_f > M_p + T, \tag{11}$$

where M_f and M_p are the monetary cost of the previous storage mode and new storage mode respectively. They are both calculated using the read frequency provided by *Predictor*.

Eq. (11) is impacted by the time period t. Since the storage cost is storing a file of size S for a time period t and c_r is the read count during t, we should set t first in order to calculate M_f and M_p. So, Eq. (11) means the new storage mode will earn back the transition cost within the time period t (t equals 30 days in our experiments). We implement the transition for each one month, which also equals to the time period t.

We calculate the storage mode for each file using its predicted read frequency in the time interval t. If the storage mode is different from the previous one and it meets Eq. (11), we change the storage mode of this file.

The storage mode table can be calculated in advance, because it is only affected by the available clouds, their pricing policies, and availabilities. When deciding the storage mode for each file, we use the read frequency and the size of the file to look up the table for the corresponding storage mode. This table is re-calculated through Algorithm 1, only when availabilities and prices are modified, some clouds are kicked out due to performance issue, or new available clouds emerge. And the new table will be input into Algorithm 2 to accommodate these situations. Algorithm 2 shows the detailed transition process.

Algorithm 2. Storage Mode Transition Process

Input: the generated table Γ, the ith file's current storage mode $M[i]$, current read frequency $R[i]$, file size $S[i]$
Output: void

```
1  dSize ← the size dimension of Γ
2  dRead ← the read frequency dimension of Γ
3  for each file i do
4    for j in len(dSize) do
5      if S[i] ≥ dSize[j] then
6        dS ← j
7      else
8        break
9      end
10   end
11   for j in len(dRead) do
12     if R[i] ≥ dRead[j] then
13       dR ← j
14     else
15       break
16     end
17   end
18   if M[i] ≠ Γ[dS][dR] then
19     T ← monetary cost of transiting from M[i]
20     if M[i] > Γ[dS][dR] + T then
21       transit from M[i] to Γ[dS][dR]
22     end
23   end
24 end
```

5.2 Complexity

Here we analyze the computational complexity of this algorithm. The two loops in lines 4 and 11 are used to look up the table, the complexity of which can be approximately considered constant, since the table is small and has only limited number of values in each dimension. Specifically, since the table is split into several pieces, we only need to find out which piece the file belongs to. Transition cost in line 19 can also be calculated in constant time.

Thus, the complexity of this algorithm is mainly the first loop, and the worst case complexity is $O(F_n)$, where F_n is the number of files. In order to reduce the complexity further, we can classify files with similar access patterns into groups, and implement transition in the unit of group. This is out of the scope of this paper.

6 Discussion

6.1 Performance of Multi-Cloud

Lots of data centers are distributed around the world, and one region such as America, Asia, usually has several data centers belonging to the same or different cloud providers. So technically all the data centers can be access by a user in a certain region, but the user would experience different performance. The latency of some data centers is very low while that of some ones may be intolerable high. CHARM chooses clouds for storing data from all the available clouds which meet the performance requirement, that is, they can offer acceptable throughput and latency when they are not in outage.

The storage mode transition does not impact the performance of the service. Since it is not a latency-sensitive process, we can decrease the priority of transition operations, and implement the transition in batch when the proxy has low workload.

6.2 Service Level Agreement and Auditing

CHARM uses the availabilities declared in the SLAs of cloud services. However, SLA does not represent the real Mean Time Between Failures (MTBF) of the cloud service, that is, it does not represent the availability of the system directly. Violating SLA is allowed, and cloud vendors only need to pay "service credits" for the violation. So, for cloud vendors, there is a tradeoff between SLA and payment. In order to test the real availability, some works [31] propose approaches to audit the real performance of clouds. Some of them require the coordination of cloud vendors, such as providing specific service API, to verify the real availability. CHARM can also rely on the performance data from third-party auditing to make storage decision. A work in [32] takes the first step to systematically detect correlations of clouds. We can take the output of their model into consideration to select the available clouds, for example, discarding the clouds which have high correlations.

6.3 Concern of Erasure Coding

The computational complexity of erasure coding is one of the most significant concerns, because it needs to implement lots of multiplication operations in Galois Field. The CPU resource may become the bottleneck of the applications and services which apply erasure coding. Recently, however, this

is not the case for erasure coding scenarios any more, since we can leverage Intel SIMD Instructions to greatly increase the coding speed (the multiplication speed is as high as 8 GB/s using commodity CPU such as Intel Core i7-3770) [26]. Thus, coding complexity can be addressed easily using commodity CPUs, making erasure coding more popular in storage systems [33], [34]. Moreover, we give n an upper limit to guarantee high performance as described in Section 4.3, which reduces the computational overhead further. How to set the upper limit is the problem that the real system developers have to deal with through real world measurements.

6.4 Other System Concerns

As a holistic storage system, there are several other factors to be considered, such as cache strategies, geographical data consistency, etc. However, we only focus on the data hosting strategy to minimize monetary cost while meeting flexible availability requirements. Though we have considered the complexity and feasibility when designing this strategy, the system design is out of the scope of this paper, and we put the detailed system design of multi-cloud data hosting into future work.

7 Evaluation

We conduct extensive simulations to evaluate the performance of our scheme. The simulations are driven by two typical real-world traces. We first briefly introduce the two collected traces and present the evaluating methodology, then show the performance of our scheme. At last, to make the results more convincing, we also implement the prototype experiments on top of four mainstream commercial clouds, the results of which prove the correctness of the simulations and the efficacy of CHARM.

7.1 Datasets

The two traces are collected from AmazingStore [20] and Corsair [21]. AmazingStore is a popular file storing and sharing platform in China. It has been deployed and maintained since April 2009, and has 10K log-in users everyday. The files in this system are mainly music and video. Corsair is a cloud storage system deployed at Tsinghua University, China. There had been already 19,892 registered users and 17.5 TB of data by September 2010. The files stored in this system have diverse types. We collected the trace of AmazingStore from January 1, 2012 to July 15, 2013 from four main servers. For Corsair the trace is collected from March to July 2010. Each line of the traces is a file access record which includes timestamp, file name, file size, and operation type (e.g., GET, PUT). The detailed properties of the two traces are shown in Table 4.

We use 15 clouds in the experiments, and they all meet the requirement of performance. The prices of these clouds are configured referring to the prices of current famous clouds (e.g., Amazon S3, Windows Azure) and their data centers. We set the clouds' availability in the interval of $[99.5, 99.95\ \text{percent}]$.

7.2 Methodology

We split the traces into pieces with the same time interval which is 30 days in our experiments. The pieces are put into CHARM one by one. CHARM reads the piece of the trace to get the files' size and current read count. Then it decides the storage mode, and calculates monetary cost for each file. We set the upper limit ξ to be 9 in CHARM.

TABLE 4
Key Facts of the Datasets

Items	AmazingStore	Corsair
Time span (Days)	575	122
No. of files	130,244	934,831
Total size (GB)	43,222	5,638
Avg file size (MB)	339.8	6.18
No. of read requests	5,209,493	1,924,451

We compare our scheme with four different data hosting schemes as described below:

RepRa [22]. Pure replication is applied in this scheme. It selects clouds randomly from all the available clouds until the selected clouds meet the required availability. Read access is directed to the cloud with the lowest read cost.

RepGr [23]. It also uses pure replication, however, it greedily selects clouds with low bandwidth price until the selected clouds meet the required availability. Read access is tackled the same as RepRa.

EraRa [16]. Pure erasure code is applied in this scheme. It sets $n = 9$, nine clouds are chosen from all the 15 clouds randomly. Then it tries m from n to 1 until it finds the value that meets the availability requirement.

EraGr [24]. Similar to EraRa, it also sets $n = 9$. Firstly, all the 15 clouds are sorted by storage price from low to high. It chooses the first nine clouds. Then it tries m from n to 1 to find the value that first meets the availability requirement.

These four schemes tackle the traces, and calculate monetary cost in the same way with CHARM.

7.3 Storage Mode Table

We generate the storage mode table based on the 15 clouds guaranteeing 99.9999 percent availability. We use different file sizes varying from 1 KB to 1 GB and different read counts varying from 0 to 100 with the step of 0.1 to calculate their corresponding storage modes (using Algorithm 1).

We get four different storage modes as shown in Fig. 4 with gray levels from 1 to 4. We only plot the read count from 0 to 3, because the storage modes are the same (i.e., gray level 4) for the read count larger than 3 no matter how much the file's size is. When the file's size is larger than 1 MB, the storage modes have explicit vertical boundaries with different read counts. That means, for large files, read count is the key to impact the storage mode.

When the file's size drops below 1 MB, the operation cost has more and more impact on the total cost. High read frequency (generating high bandwidth cost) gives advantages to replication mechanism (i.e., $m = 1$). So, similarly, high operation cost also gives advantages to replication mechanism when the file's size is small. That is why gray level 4 puts its feet into the region of lower read count and smaller file size. This storage mode table only depends on prices of the available clouds and required availability. If the prices change, the table will change accordingly, becoming a different one.

TABLE 5
Monetary Cost of Different Schemes for AmazingStore Trace (Thousand $)

Availability	RepRa	RepGr	EraRa	EraGr	CHARM
99.99%	383.18	239.38	285.55	277.71	210.09
99.999%	377.38	239.38	281.31	275.25	212.64
99.9999%	283.68	239.38	270.14	275.25	217.19
99.99999%	453.29	288.71	269.67	275.25	222.08

7.4 Monetary Cost

We set different availability levels from 99.99 to 99.99999 percent, and run the two traces applying the five schemes respectively. The total cost of CHARM includes storage/bandwidth/operation costs and transition cost. The results of AmazingStore trace are shown in Table 5. Since the read count of files in AmazingStore trace is high (i.e., 39.9 on average in 575 days), RepGr is better than EraGr except the highest availability case. In order to guarantee high availability, RepGr has to store more replicas whose storage cost exceeds the saving on bandwidth. The cost of EraGr for 99.99 percent is higher than that in higher availability, because EraGr has to reduce m to get higher availability, and it happens to exclude the cloud with higher bandwidth cost. CHARM has the lowest cost, it reduces about 9.3-23.1 percent compared to RepGr, and reduces about 19.3-24.3 percent compared to EraGr.

From the detailed monetary cost as shown in Table 7, we can see that CHARM spends a little more storage cost to achieve much lower bandwidth cost. The detailed monetary cost of other availability levels shows similar results. RepRa and EraRa select clouds randomly, so the cost does not show strictly increase with the increase of availability.

The results of Corsair trace are shown in Table 6. In this case EraGr is better than RepGr, because the read frequency of Corsair is relatively low, only about 2.1 on average in 122 days. For RepGr, the saving on bandwidth cannot afford high storage cost. From Table 8, we can clearly see that RepGr saves 37 percent on bandwidth cost but spends 138 percent additional storage cost compared to EraGr. CHARM is still the best one in the five schemes. It reduces about 23.3-43.4 percent of the cost compared to RepGr, and reduces about 6.1-10.4 percent of the cost compared to EraGr.

In our experiments, the storage mode transition is implemented 80,062 times with 719.74 dollars transition cost for AmazingStore trace with 99.99999 percent availability. While for Corsair trace the numbers are 11,893 times and 1.9 dollars. This shows that we use low transition cost to cost-efficiently re-distribute data.

7.5 Tolerating Price Adjustment

We adjust the price of clouds based on the fact that cloud providers have adjusted the price for several times [7], [8].

TABLE 6
Monetary Cost of Different Schemes for Corsair Trace($)

Availability	RepRa	RepGr	EraRa	EraGr	CHARM
99.99%	4824.00	4433.20	3755.43	3449.80	3139.79
99.999%	6413.49	4433.20	3881.68	3639.62	3260.59
99.9999%	6069.24	4433.20	4081.91	3639.62	3383.61
99.99999%	6206.31	6036.55	3901.28	3639.62	3402.34

TABLE 7
Detailed Monetary Cost of AmazingStore Trace for 99.99999 percent Availability ($)

Items	RepRa	RepGr	EraRa	EraGr	CHARM
storage	158418	156677	67983	65655	77958
bandwidth	294867	132030	201681	209571	144118
operation	2.08	0	9.91	21.05	0.7699
total	453287	288707	269674	275248	222077

So when the simulation runs half of the traces (i.e., nine months for AmazingStore, and two months for Corsair), we decrease and increase the price by 50 percent respectively to simulate the situation. Moreover, the prices that are modified include storage, bandwidth, and operation.

The cloud we choose for price decrease is not used by the five schemes. That is to say this cloud has relatively higher price before price adjustment. The result of 99.999 percent availability is shown in Table 9, other experiments have similar results. When the price adjustment occurs, CHARM re-calculates the storage mode table, and uses the new table to store data and implement transition. So it shows great adaptability compared with the other four schemes which cannot change the clouds dynamically according to the price fluctuation. CHARM saves 23.4 and 33.4 percent monetary cost compared to RepGr and EraGr respectively for AmazingStore trace. The Corsair trace shows similar results of 28.9 and 13.4 percent.

We increase the price of the cloud that is used by all the five schemes. If the price is high enough, CHARM may exclude this cloud, and transfer the data on this cloud to other clouds. However, the other schemes have to bear the price adjustment. The result shows CHARM cuts 16.6 and 25.4 percent monetary cost compared to RepGr and EraGr respectively for AmazingStore trace. Similarly, for Corsair trace, the savings are 35.2 and 13.3 percent respectively.

7.6 Supporting Varying Availability

Different types of data may require different availabilities [35]. For example, backup data usually requires relatively low availability, while documents in work folders demand high availability. The experiments in Section 7.4 prove the effectiveness of CHARM in this scenario since it performs best for various availabilities.

However, a more complicated use case is that the availability is varying with the access frequency of data. For example, "hot" data may demand high availability while "cold" data does not have that strict requirement. In order to show that CHARM can also naturally adapt to this scenario, we run the two traces and assign different availabilities to the files according to their access frequency. More

TABLE 8
Detailed Monetary Cost of Corsair Trace for 99.99999 percent Availability ($)

Items	RepRa	RepGr	EraRa	EraGr	CHARM
storage	5262.77	5093.00	2263.56	2134.21	2215.50
bandwidth	943.55	943.55	1633.99	1497.70	1186.30
operation	0	0	3.74	7.71	0.55
total	6206.32	6036.55	3901.28	3639.61	3402.35

TABLE 9
Monetary Cost of Different Schemes with Price Fluctuation ($)

Datasets	Price fluctuation	RepRa	RepGr	EraRa	EraGr	CHARM
AmazingStore	50%	279913.11	239382.75	300459.22	275247.57	183374.50
	150%	387825.87	255135.52	299536.90	285192.69	212821.77
Corsair	50%	5125.73	4433.20	3671.78	3639.61	3152.10
	150%	4693.39	5051.27	4184.60	3777.82	3274.46

specifically, in our experiments three read requests a month is the boundary between high (99.99999 percent) and low (99.99 percent) availabilities. In order to avoid switching back and forth frequently, when the frequency drops below one request a month we change the availability from high to low, and when the frequency rises above five requests a month, the availability is changed from low to high. Setting these threshold values is reasonable since there is no strict boundary between different availabilities. The other schemes also use the same way to switch the availability.

The results are shown in Table 10, and we omit the results of RepRa and EraRa. CHARM outperforms RepGr and EraGr by 13.8 and 21.0 percent savings respectively for AmazongStore, and by 30.0 and 9.8 percent savings for Corsair. If we compare the results with the numbers in Tables 7 and 8, we can see that the storage cost decreases a lot since a proportion of files are stored using lower availability, and the bandwidth cost only increases a little. The switch cost of RepGr is much lower than EraGr, because RepGr uses only DELETE operation to switch from high to low availability, while EraGr needs to GET the file first and PUT the file using the storage mode of low availability.

The availability of clouds can also be changed, for instance, from low to high due to the upgrade of devices, or from high to low due to excessive operating cost of cloud vendors. Thus, we run experiments to simulate these scenarios. We reduce the availabilities of two clouds that are used by the three schemes to 90.0 percent. The result is that CHARM outperforms other schemes by 10.3-44.0 percent, revealing flexible adaptation. We do not show detailed numbers due to the limited space.

7.7 Data Distribution

To reveal more details about data placement, we show the data distribution in clouds in Fig. 5. It is the result of 99.99999 percent availability, other experiments have similar data distribution. AmazingStore trace has high read frequency, CHARM uses storage mode with $m = 1$ and $n = 3$ (i.e., replication) to store "hot" files. If "hot" files become "cold", their storage modes are changed to the storage modes with higher m and n.

As shown in Fig. 5a, there are three clouds storing more data than the other six clouds to serve "hot" data access. There are two obvious decrease points of "CHARM-3", which mean that many files become "cold" ones and their storage modes are changed. The vertical coordinate is log scale. We can see that RepGr places all the data in three clouds, storage usage of which is much higher than that of CHARM and EraGr. EraGr uniformly distributes data among 9 clouds. Besides getting the lowest monetary cost, CHARM is prone to distribute data evenly, which efficiently reduces the vendor lock-in risk. The data distribution advantage of CHARM can be seen more clearly in Fig. 5b. Since there is much less "hot" data compared to AmazingStore trace, the three clouds in CHARM for serving "hot" data do not store much more data than the other six clouds. The data distribution of CHARM is very similar to that of EraGr.

TABLE 10
Monetary Cost of Different Schemes with Varying Availability of Files ($)

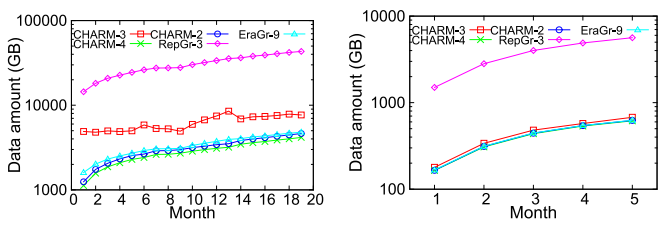
Datasets	Items	RepGr	EraGr	CHARM
AmazingStore	storage	118133.1	59241.9	71679.9
	bandwidth	132652.7	214479.2	144487.5
	operation	0	21.289	0.75109
	switch	622.7	3752.8	1141.4
	total	250785.8	273742.4	216168.1
Corsair	storage	3553.8	1878.1	1957.5
	bandwidth	944.0	1606.6	1192.9
	operation	0	8.2113	0.4662
	switch	0.459	55.13	2.34
	total	4497.8	3492.9	3150.8

"switch" means the monetary cost generated by switching availability requirements.

7.8 Applying to Complex Request Pattern

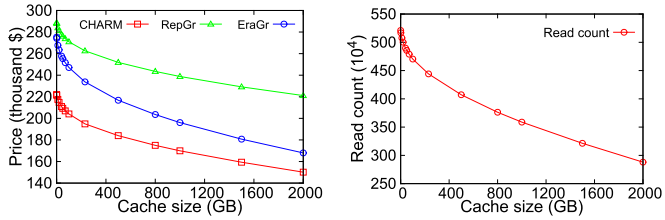
Clearly, RepGr usually performs better for AmazingStore trace while EraGr performs better for Corsair trace. CHARM combines the merits of the two schemes to achieve the best performance, since it picks different storage modes for the files with different access frequency, which determines great adaptation.

Cache is a commonly used technique to relieve the burden of back-end storage, shaping the data access pattern that is actually served by the back-end storage. Since AmazingStore trace has higher read frequency, we use a cache to filter the trace to show that CHARM well applies to various



(a) AmazingStore trace (b) Corsair trace

Fig. 5. Data distribution of the two traces. The name in the legend is composed of two parts, the first part is the name of the scheme and the second part is the number of the clouds that store the same amount of data. For example, "RepGr-3" means RepGr uses three clouds and they store the same amount of data, the line in the figure shows the amount of data in one cloud. CHARM uses nine clouds with three different storage usages.



(a) Monetary cost (b) Read count

Fig. 6. Results of different cache sizes. The left one is total monetary cost of CHARM, RepGr and EraGr on different cache sizes. The right one is total read count of filtered traces with different cache sizes.

access patterns. More specifically, we use LRU for this cache with the cache size varying from 1 to 2,000 GB. Fig. 6b shows the total number of requests received by the back-end storage after filtered by the cache. With the increase of the cache, read count drops quickly. Then we apply the five schemes to the filtered traces with 99.99999 percent availability.

The total costs of different schemes are shown in Fig. 6a (RepRa and EraRa are not plotted because their costs are also affected by the randomly selected clouds). When the cache size is small, the files still have high read frequency, and many files change from "hot" to "cold". CHARM chooses the storage mode with low bandwidth cost for "hot" files, and chooses the one with low storage cost for "cold" files, and also implements necessary storage mode transition. So, it performs much better than RepGr and EraGr.

When cache size increases, requests that go to back-end clouds decrease, making the proportion of bandwidth cost much smaller. So RepGr gradually loses its advantage. On the contrary, EraGr performs better due to its advantage on storage cost, approaching the performance that CHARM achieves. But some files still have relatively high access frequency, and EraGr cannot get the optimal m and n in some cases. That is why CHARM still performs better than EraGr when the cache is large. This experiment proves that CHARM effectively adapts to diverse access patterns, and achieves the best performance.

7.9 Prototype Experiments

We implemented the prototype experiments on four mainstream commercial clouds: Amazon S3, Windows Azure, Google Cloud Storage and Aliyun OSS, and pick 10 different data centers[2] from them. We created accounts in the four clouds, and replayed AmazingStore trace and Corsair trace for a whole month, using three different schemes (i.e., CHARM, RepGr, EraGr) which choose their preferred data centers from the 10 available ones.

In order to make sure the experiments can be finished in one month, we scale down the traces on the premise of not affecting the correctness. That is, we randomly select 1 percent files from the two traces, gathering their access records into two small-scale traces. Since the files in AmazingStore is large (e.g., hundreds of MB, several GB), we make them 10x smaller, which also does not impact the experiment effect. We replay the 10th month of AmazingStore trace and the 3rd month of Corsair trace. Before the implementation, we put the files that already exist before the two time points into clouds. Then we replay the traces of the specified months. We also run simulations for the same traces to contrast with the real-world experiments.

The detailed results are shown in Table 11. For the prototype experiments, CHARM outperforms RepGr and EraGr by 21.7 and 27.8 percent for AmazingStore trace, and similarly the savings are 37.8 and 13.8 percent for Corsair trace, which proves the efficacy of CHARM. Equally importantly, the prototype experiments show similar results as the simulation results, which proves the correctness of our simulations.

There are two interesting discrepancies between the prototype experiments and the simulations. The one is that the total cost generated in the prototype experiments is around \$0.1 less than that in the simulations for AmazingStore trace, and the difference is about \$0.3 for Corsair trace. There are two reasons for this. Firstly, according to our observation, different vendors apply diverse statistic methods and use different precision to compute the resource usage of customers. Secondly, cloud vendors are prone to round down the amount of resource usage, making the real cost a little smaller.

The other one is that operation cost in the prototype experiments is higher than that in the simulations. This is because Google Cloud Storage presents about 1x more transactions than the number of GET operations we generated, and Windows Azure presents about 5x more transactions. Maybe there are some types of transactions that should be issued together with GET operation. Since the proportion of operation cost is very small, the total cost is not impacted.

8 Related Work

With the blossom of cloud services [36], there is a recent interest in addressing how to migrate data and applications into clouds seamlessly [37], [38]. The system designed in [37] migrates Network File System (NFS) into the cloud, and meanwhile makes it feel like working locally. A similar work in [38] proposes a hybrid cloud-based deployment, where enterprise operations are partly hosted on-premise and partly in the cloud.

Lots of works optimize the performance of the services from diverse aspects. The SCADS Director [39] reconfigures the storage system on-the-fly, while guaranteeing strict performance service-level objectives (SLOs) expressed using upper percentiles of request latency. Scc [40] addresses how to select the best combination of diverse storage devices (e.g., disks, SSDs, DRAM) to minimize cluster storage cost. Some works focus on the selection of clouds, ISPs, and servers [22], [23], [41]. For example, [23] focuses on placing content using multiple content distribution networks to optimize cost and performance. DONAR [22] is a distributed system which uses a distributed algorithm to direct client requests to a particular replica, based on performance, load, and cost. DREAM [41] focuses on VoD applications.

2. Amazon S3: Standard, Northern California, Singapore, Tokyo. Windows Azure: West US. Google Cloud Storage: America, Asia. Aliyun OSS: Beijing, Qingdao, HongKong. The expense is generated using the prices in 2014.10.22-2014.11.21. We skip the first free tie of Amazon S3 (i.e., the first 1 GB outgoing bandwidth).

TABLE 11
Detailed Monetary Cost of the Prototype Experiments and Simulations for 99.99999 percent Availability ($)

Items	Amazingstore						Corsair					
	RepGr		EraGr		CHARM		RepGr		EraGr		CHARM	
	P	S	P	S	P	S	P	S	P	S	P	S
Bandwidth	4.4778	4.4805	7.2530	7.2982	4.5413	4.6216	2.3054	2.4324	3.6985	3.9622	2.7062	2.9722
Storage	3.4102	3.4198	1.2945	1.2982	1.6356	1.6394	6.0018	6.1505	2.2784	2.3348	2.4583	2.5219
Operation	0.0004	0.0001	0.0130	0.0059	0.0010	0.0005	0.0004	0.0001	0.0204	0.0132	0.0058	0.0036
Total	7.8884	7.9004	8.5605	8.6023	6.1779	6.2614	8.3077	8.5830	5.9973	6.3101	5.1703	5.4977

"P" means prototype experiment While "S" means simulation result.

There is a common concern that moving data into a single could would incur vendor lock-in risk. So many works propose storage architectures and mechanisms based on multiple clouds. DepSky [15] stores data, even critical data (e.g., medical record databases, financial data), into multiple clouds, guaranteeing data availability and security. NCCloud [42] designs a coding scheme to recover data with less bandwidth consumption when a cloud busts. To reduce monetary cost, RACS [16] applies the RAID-like technique at the cloud storage level, which avoids vendor lock-in efficiently and reduces the cost of switching cloud vendors. Scalia [24] is inspired by RACS, and moves a step further that taking data access pattern into consideration, but it does not consider different redundancy mechanisms.

Moreover, for convenient deployment of multi-cloud services, many new tools are developed, such as Apache Libcloud [19] which provides a unified interface above different clouds. New types of vendors, such as DuraCloud [17], Cloud Foundry [18], have emerged and been growing rapidly. DuraCloud provides a convenient service to move content copies into the cloud, and store them with several different providers, all with just one click. Cloud Foundry is an open platform as a service, providing a choice of clouds, developer frameworks, and application services. It becomes much faster and easier to build, test, deploy, and scale applications.

There are two works about the comparison between erasure coding and replication in Peer-to-Peer systems [27], [28]. However, no work compares these two mechanisms in multi-cloud environment, which is proved very different from the results in the two works according to our analysis in Section 3. On the other hand, data hosting in Grid/Peer-to-Peer storage systems has been researched a lot [43], [44], [45], [46]. A prominent property of these storage systems is that the storage nodes are unstable or anomalous. However, this is not the case for multi-cloud environment, which makes it difficult and unbefitting to deploy those data hosting schemes.

A similar work to ours is CAROM [47], which replication and erasure coding in multiple data centers. But it does not consider the heterogeneity of multi-cloud and the selection of clouds. They design a cache in the primary data center. When a file is swapped out, this file is stored using erasure coding across multiple data centers. When this file is accessed again, it will be stored back to the cache. This scheme is efficient for the trace used in their paper. However, its performance relies on the characters of the targeted trace, more specifically, the cache hit rate (about 90 percent for their trace). For AmazingStore trace, the hit rate is only 44.7 percent with the cache size of 2 TB (see Fig. 6b). Frequent data swap inevitably induces much additional monetary cost to CAROM, which makes it even not competitive compared with the greedy data hosting schemes.

The data hosting schemes mentioned above focus on different aspects, e.g., optimizing performance, avoiding vender lock-in. CHARM, however, appeals to the particularities of multi-cloud environment, and hosts data into multiple clouds cost-effectively, while guaranteeing flexible availability and avoiding vender lock-in.

9 Conclusion

Cloud services are experiencing rapid development and the services based on multi-cloud also become prevailing. One of the most concerns, when moving services into clouds, is capital expenditure. So, in this paper, we design a novel storage scheme CHARM, which guides customers to distribute data among clouds cost-effectively. CHARM makes fine-grained decisions about which storage mode to use and which clouds to place data in. The evaluation proves the efficiency of CHARM.

Acknowledgments

The work was supported by National Basic Research Program of China ("973") under Grant 2011CB302305, High-Tech Research and Development Program of China ("863 China Cloud" Major Program) under Grant 2015AA01A201, China NSF under Grants 61232004 and 61471217, and China Postdoctoral Science Fund under Grant 2014M550735. The authors also thank Songbin Liu for the Corsair trace.

References

[1] Aliyun OSS (Open Storage Service). [Online]. Available: http://www.aliyun.com/product/oss, 2014.

[2] Gartner: Top 10 cloud storage providers. [Online]. Available: http://www.networkworld.com/news/2013/010313-gartner-cloud-storage-265459.html?page=1, 2013.

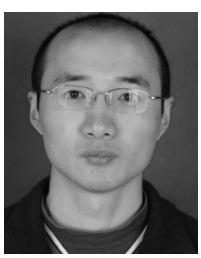
[3] Z. Li, C. Jin, T. Xu, C. Wilson, Y. Liu, L. Cheng, Y. Liu, Y. Dai, and Z.-L. Zhang, "Towards network-level efficiency for cloud storage services," in *Proc. ACM SIGCOMM Internet Meas. Conf.*, 2014, pp. 115–128.

[4] Z. Li, C. Wilson, Z. Jiang, Y. Liu, B. Y. Zhao, C. Jin, Z.-L. Zhang, and Y. Dai, "Efficient batched synchronization in dropbox-like cloud storage services," in *Proc.ACM/IFIP/USENIX 14th Int. Middleware Conf.*, 2013, pp. 307–327.

[5] C. M. M. Erin Allen, "Library of congress and duracloud launch pilot program using cloud technologies to test perpetual access to digital content," *Library Cong., News Releases*. [Online]. Available: http://www.loc.gov/today/pr/2009/09-140.html, 2009.

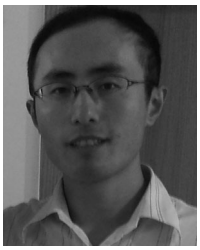
[6] A. Li, X. Yang, S. Kandula, and M. Zhang, "CloudCmp: Comparing public cloud providers," in *Proc. ACM SIGCOMM Internet Meas. Conf.*, 2010, pp. 1–14.

[7] Windows Azure Pricing Updates. [Online]. Available: http://azure.microsoft.com/en-us/updates/azure-pricing-updates/, 2014.
[8] Google Cloud Platform Pricing Updates. [Online]. Available: http://googlecloudplatform.blogspot.com/2014/03/google-cloud-platform-live-blending-iaas-and-paas-moores-law-for-the-cloud.html, 2014.
[9] It's Official, the Nirvanix Cloud Storage Service is Shutting Down. [Online]. Available: http://techcrunch.com/2013/09/27/its-official-the-nirvanix-cloud-storage-service-is-shutting-down/, 2013.
[10] Shutting Down Ubuntu One File Services. [Online]. Available: http://blog.canonical.com/2014/04/02/shutting-down-ubuntu-one-file-services/, 2014.
[11] Nirvanix Provides Cautionary Tale for Cloud Storage. [Online]. Available: http://www.forbes.com/sites/tomcoughlin/2013/09/30/nirvanix-provides-cautionary-tail-for-cloud-storage/, 2013.
[12] Google Outages Damage Cloud Credibility. [Online]. Available: https://www.networkworld.com/news/2009/092409-google-outages-damage-cloud.html, 2009.
[13] Rackspace to issue as much as $3.5M in customer credits after outage. [Online]. Available: http://www.networkworld.com/news/2009/070609-rackspace-outage.html, 2009.
[14] Summary of the amazon EC2 and amazon RDS service disruption in the US east region. [Online]. Available: http://aws.amazon.com/cn/message/65648/, 2011.
[15] A. Bessani, M. Correia, B. Quaresma, F. André, and P. Sousa, "DepSky: Dependable and secure storage in a cloud-of-clouds," in *Proc. 6th Conf. Comput. Syst.*, 2013, pp. 31–46.
[16] H. Abu-Libdeh, L. Princehouse, and H. Weatherspoon, "RACS: A case for cloud storage diversity," in *Proc. 1st ACM Symp. Cloud Comput.*, 2010, pp. 229–240.
[17] DuraCloud. [Online]. Available: http://www.duracloud.org/, 2014.
[18] Cloud Foundry. [Online]. Available: http://www.cloudfoundry.org/, 2014.
[19] Apache Libcloud. [Online]. Available: http://libcloud.apache.org/, 2014.
[20] AmazingStore. [Online]. Available: http://cn.amazingstore.org/, 2014.
[21] S. Liu, X. Huang, H. Fu, and G. Yang, "Understanding data characteristics and access patterns in a cloud storage system," in *Proc. 13th IEEE/ACM Int. Symp. Cluster, Cloud, Grid Comput.*, 2013, pp. 327–334.
[22] P. Wendell, J. W. Jiang, M. J. Freedman, and J. Rexford, "Donar: Decentralized server selection for cloud services," in *Proc. ACM SIGCOMM Conf.*, 2010, pp. 231–242.
[23] H. H. Liu, Y. Wang, Y. R. Yang, H. Wang, and C. Tian, "Optimizing cost and performance for content multihoming," in *Proc. ACM SIGCOMM Conf. Appl., Technol., Archit., Protocols Comput. Commun.*, 2012, pp. 371–382.
[24] T. G. Papaioannou, N. Bonvin, and K. Aberer, "Scalia: An adaptive scheme for efficient multi-cloud storage," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage, Anal.*, 2012, p. 20.
[25] J. S. Plank, "Erasure codes for storage systems: A brief primer," *Usenix Mag.*, vol. 38, no. 6, pp. 44–50, 2013.
[26] J. S. Plank, K. M. Greenan, and E. L. Miller, "Screaming fast galois field arithmetic using Intel SIMD instructions," in *Proc. 11th USENIX Conf. File Storage Technol.*, 2013, pp. 299–306.
[27] H. Weatherspoon and J. D. Kubiatowicz, "Erasure coding vs. replication: A quantitative comparison," in *Proc. 1st Int. Workshop Peer-to-Peer Syst.*, 2002, pp. 328–338.
[28] R. Rodrigues and B. Liskov, "High availability in DHTs: Erasure coding vs. replication," in *Proc. 4th Int. Conf. Peer-to-Peer Syst.*, 2005, pp. 226–239.
[29] I. M. Bomze, M. Budinich, P. M. Pardalos, and M. Pelillo, "The maximum clique problem," in *Handbook of Combinatorial Optimization*. New York, NY, USA: Springer, 1999, pp. 1–74.
[30] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Syst. Tech. J.*, vol. 49, no. 2, pp. 291–307, 1970.
[31] M. A. Shah, M. Baker, J. C. Mogul, and R. Swaminathan, "Auditing to keep online storage services honest," in *Proc. 11th USENIX Workshop Hot Topics Oper. Syst.*, 2007, p. 11.
[32] E. Zhai, R. Chen, D. I. Wolinsky, and B. Ford, "Heading off correlated failures through independence-as-a-service," in *Proc. 11th USENIX Conf. Oper. Syst. Des., Implementation*, 2014, pp. 317–334.
[33] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin, "Erasure coding in windows azure storage," in *Proc. USENIX Annu. Tech. Conf.*, 2012, p. 2.
[34] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur, "XORing elephants: novel erasure codes for big data," *VLDB Endowment*, vol. 6, no. 5, pp. 325–336, 2013.
[35] X. Ma, "On the feasibility of data loss insurance for personal cloud storage," presented at the 6th USENIX Workshop Hot Topics Storage File Syst., Philadelphia, PA, USA, 2014.
[36] K. Zhou, H. Wang, and C. Li, "Cloud storage technology and its applications," *ZTE Commun.*, vol. 8, no. 4, pp. 27–30, 2010.
[37] M. Vrable, S. Savage, and G. M. Voelker, "BlueSky: A cloud-backed file system for the enterprise," in *Proc. USENIX Conf.File Storage Technol.*, 2012, p. 19.
[38] M. Hajjat, X. Sun, Y.-W. E. Sung, D. Maltz, S. Rao, K. Sripanidkulchai, and M. Tawarmalani, "Cloudward bound: Planning for beneficial migration of enterprise applications to the cloud," in *Proc. ACM SIGCOMM Conf.*, 2010, pp. 243–254.
[39] B. Trushkowsky, P. Bodík, A. Fox, M. J. Franklin, M. I. Jordan, and D. A. Patterson, "The SCADS director: Scaling a distributed storage system under stringent performance requirements," in *Proc. 9th USENIX Conf. File, Stroage Technol.*, 2011, p. 12.
[40] H. V. Madhyastha, J. C. McCullough, G. Porter, R. Kapoor, S. Savage, A. C. Snoeren, and A. Vahdat, "scc: Cluster storage provisioning informed by application characteristics and SLAs," in *Proc. USENIX Conf. File, Stroage Technol.*, 2012, p. 23.
[41] Y. Zhao, H. Jiang, K. Zhou, Z. Huang, and P. Huang, "Meeting service level agreement cost-effectively for video-on-demand applications in the cloud," in *Proc. IEEE INFOCOM*, 2014, pp. 298–306.
[42] Y. Hu, H. C. Chen, P. P. Lee, and Y. Tang, "NCCloud: Applying network coding for the storage repair in a cloud-of-clouds," in *Proc. 10th USENIX Conf. File, Stroage Technol.*, 2012, p. 21.
[43] M. Pitkanen, R. Moussa, M. Swany, and T. Niemi, "Erasure codes for increasing the availability of grid data storage," in *Proc. Int. Conf. Internet Web Appl. Serv./Adv. Int. Conf. Telecommun.*, 2006, p. 185.
[44] H. B. Ribeiro and E. Anceaume, "Datacube: A P2P persistent data storage architecture based on hybrid redundancy schema," in *Proc. 18th Euromicro Int. Conf. Parallel, Distrib. Netw.-Based Process.*, 2010, pp. 302–306.
[45] A. Duminuco and E. W. Biersack, "Hierarchical codes: A flexible trade-off for erasure codes in peer-to-peer storage systems," *Peer-to-Peer Netw. Appl.*, vol. 3, no. 1, pp. 52–66, 2010.
[46] W. Lin, D. Chiu, and Y. Lee, "Erasure code replication revisited," in *Proc. 4th Int. Conf. Peer-to-Peer Comput.*, 2004, pp. 90–97.
[47] Y. Ma, T. Nandagopal, K. P. Puttaswamy, and S. Banerjee, "An ensemble of replication and erasure codes for cloud file systems," in *Proc. IEEE INFOCOM*, 2013, pp. 1276–1284.

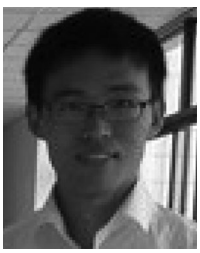


Quanlu Zhang received the BSc degree from the Harbin Institute of Technology in 2012. He is currently working toward the PhD degree at Peking University, Beijing, China. His current research areas mainly consist of cloud computing/storage, data center storage, file system design, power saving, and resource management.

Shenglong Li received the BSc degree from Tianjin University in 2013. He is currently working toward the PhD degree at Peking University, Beijing, China. His current research areas mainly consist of cloud computing/storage, cloud synchronization techniques, resource allocation, and SSD-based storage system.



Zhenhua Li received the PhD degree from the School of EECS, Peking University. He is currently a post doc at the School of Software, Tsinghua University, Beijing, China, and an associate researcher in TNLIST, Beijing. His current research areas mainly consist of cloud computing/storage and mobile/cellular Internet. He has published one book and more than 30 papers in the above areas. He is a member of the ACM and the IEEE.



Yuanjian Xing received the BSc degree from the Harbin Institute of Technology in 2008, and the PhD degree from Peking University in 2013. His research interests include peer-to-peer networks and wide-area systems, with an emphasis on data storage and content distribution.

Zhi Yang received the BS degree from the Harbin Institute of Technology in 2005 and the PhD degree in computer science from Peking University in 2010. He is currently an associate professor at Peking University. His research Interests include the areas of security and privacy, networked and distributed systems, and data intensive computing. Most recently, he designing systems for processing large graphs.

Yafei Dai received the PhD degree in computer science from the Harbin Institute of Technology. She is currently a professor at the Computer Science Department, Peking University. Her research areas include networked and distributed systems, P2P computing, network storage, and online social networks. She is a member of the IEEE.