# Systematic Data Placement Optimization in Multi-Cloud Storage for Complex Requirements

Maomeng Su, Lei Zhang, Yongwei Wu, *Member, IEEE*, Kang Chen, and Keqin Li, *Fellow, IEEE*

**Abstract**—Multi-cloud storage can provide better features such as availability and scalability. Current works use multiple cloud storage providers with erasure coding to achieve certain benefits including fault-tolerance improving or vendor lock-in avoiding. However, these works only use the multi-cloud storage in ad-hoc ways, and none of them considers the optimization issue in general. In fact, the key to optimize the multi-cloud storage is to effectively choose providers and erasure coding parameters. Meanwhile, the data placement should satisfy system or application developers' requirements. As developers often demand various objectives to be optimized simultaneously, such complex requirement optimization cannot be easily fulfilled by ad-hoc ways. This paper presents Triones, a systematic model to formally formulate data placement in multi-cloud storage by using erasure coding. Firstly, Triones addresses the problem of data placement optimization by applying non-linear programming and geometric space abstraction. It could satisfy complex requirements involving multi-objective optimization. Secondly, Triones can effectively balance among different objectives in optimization and is scalable to incorporate new ones. The effectiveness of the model is proved by extensive experiments on multiple cloud storage providers in the real world. For simple requirements, Triones can achieve 50 percent access latency reduction, compared with the model in $\mu$LibCloud. For complex requirements, Triones can improve fault-tolerance level by 2× and reduce access latency and vendor lock-in level by 30∼70 percent and 49.85 percent respectively with about 19.19 percent more cost, compared with the model only optimizing cost in Scalia.

**Index Terms**—Systematic model, data placement optimization, multi-cloud storage, complex requirements

## 1 Introduction

Cloud storage can provide virtually unlimited storage capacity for its users. It has been a trend that large numbers of organizations, companies, government departments, etc., are storing their data into cloud [1]. However, only using one cloud storage provider is more likely to suffer from single-point failures and vendor lock-in [2], [3]. As a result, the multi-cloud storage that relies on multiple cloud storage providers to place data at some redundance level has been used by current works [2], [3], [4], [5], [6], [7]. It can provide better service quality including vendor lock-in avoiding as well as fault-tolerance improving. These features are extremely beneficial to systems or applications such as data backup, document archiving, or electronic health recording, which need to keep a large amount of data.

In such cases, erasure coding [8], [9], [10] is usually applied for further improvement as it can significantly reduce the cost of data storage compared to full replication [11], [12], [13]. The total amount of data that must be transferred over the network can also be reduced. With erasure coding, each data object is evenly divided into $k$ blocks and then these blocks are used to generate $n-k$ encoded data blocks ($n$ blocks in total, $n>k$). This is called as parameter $(n,k)$ of erasure coding. These $n$ data blocks will be uploaded into $n$ cloud storage providers, with each provider holding just one data block respectively. Any $k$ blocks from the $n$ ones can be used to reconstruct the original data object. We can see that erasure coding can naturally fit into the multi-cloud storage.

However, many previous works [2], [4], [5], [6] only used the erasure-coding based multi-cloud storage to achieve certain features. Few of them considered the optimization issue under this circumstance. For example, both DepSky [5] and $\mu$LibCloud [6] have achieved low access latency, but they did not give a solution to achieve optimal access performance from the multi-cloud storage to a specific location. Scalia [7] has done the optimization work on the cost. However, it only conducted single objective optimization, which is far from enough for the multi-cloud storage. Multi-objective optimization [14], [15] such as optimizing cost, access latency, and fault-tolerance at the same time has been left unaddressed.

In fact, the most important part of optimizing the multi-cloud storage[1] is the optimization on data placement, which is to choose an optimized data placement configuration. A *data placement configuration* in the multi-cloud storage consists of erasure coding parameter $(n,k)$ and $n$ cloud storage providers.

- *M. Su, L. Zhang, Y. Wu, and K. Chen are with the Department of Computer Science and Technology, Tsinghua National Laboratory for Information Science and Technology (TNLIST), Tsinghua University, Beijing 100084, China, the Research Institute of Tsinghua University in Shenzhen, Shenzhen 518057, China, and the Technology Innovation Center at Yinzhou, Yangtze Delta Region Institute of Tsinghua University, Ningbo 315000, Zhejiang, China. E-mail: {smm11, lei-zhang11}@mails.tsinghua.edu.cn, {wuyw, chenkang}@tsinghua.edu.cn.*
- *K. Li is with the Department of Computer Science, State University of New York, New Paltz, NY 12561. E-mail: lik@newpaltz.edu.*

*Manuscript received 27 Nov. 2014; revised 24 June 2015; accepted 12 July 2015. Date of publication 0 . 0000; date of current version 0 . 0000.*
*Recommended for acceptance by G. Min.*
*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/TC.2015.2462821*

[1] In the rest of this paper, multi-cloud storage is used to represent the storage which consists of multiple cloud storage providers and stores data stripped by using erasure coding.

It is non-trivial to achieve data placement optimization in general in the multi-cloud storage. Data placement configurations with diverse cloud storage providers and different $(n, k)$ parameters offer totally different serving ability (e.g., cost, access latency, or vendor lock-in). Moreover, it further complicates the optimization that developers do have different optimization requirements, according to the properties of their systems or applications. Here, we use *simple requirement* to represent single objective optimization, i.e., optimizing only one factor. *Complex requirement* is used to represent the situation of optimizing multiple factors at the same time. For instance, some developers who want to store critical data in the multi-cloud storage would be interested in optimizing factors including cost, fault-tolerance level (FTL), and vendor lock-in level simultaneously. Other developers might care more about those factors of cost, access latency, as well as availability for data that are accessed frequently. Moreover, as an additional demand for the optimization model of the multi-cloud storage, it must be general enough to incorporate new factors and new requirements that might be raised in the future. This is meaningful as the cloud storage technology is still under rapid development. Ad-hoc ways of randomly choosing data placement configurations used in previous works can hardly fulfill the optimization to satisfy complex requirements. Furthermore, it is impossible for developers to deal with such complicated data placement optimization issue themselves.

In this paper, we present Triones, a systematic model to formulate data placement in the multi-cloud storage as well as the ways of its optimization. Triones applies non-linear programming [16], [17] to define the problem of data placement optimization under complex requirements. In the non-linear programming, Triones regards all the factors associated with the multi-cloud storage as derived variables from basic variables and constants. Basic variables are used as representation of the final optimized results (configurations) for data placement. They consist of a set of 1 or 0 to represent whether a corresponding provider is used or not with an additional variable $k$, representing the parameter of erasure coding. Besides, constants stand for the characteristics of underlying cloud storage providers.

The objective function of the non-linear programming is a combination of factors to be optimized. They map to a complex requirement demanded by system or application developers. Moreover, inequalities in the non-linear programming are used to represent constraints on factors. The left-hand side of each constraint inequality represents one factor under constraint while the right-hand side of it is the quantifiable constraint on this factor.

We use euclidean distance measure in an abstract geometric space [18], [19] to balance among different kinds of factors to get the optimized results for the objective function. The results correspond to the optimized data placement configurations that satisfy developers' complex requirements subject to certain constraints. In addition, Triones can also address data placement optimization under simple requirements, in which cases the objective function contains only one factor.

In this paper, we have made the following contributions:

1) We analyze data placement optimization in the multi-cloud storage and point out several aspects that complicate the process to choose optimized data placement configurations. One is that complex requirements are much common for developers who intend to deploy their systems or applications in the multi-cloud storage. The other is that new factors will be under consideration in the future. We think these issues on optimization should be considered systematically instead of in ad-hoc ways. This motivates a systematic approach to rethinking the data placement optimization problem in the multi-cloud storage.
2) We propose a systematic model called Triones. It uses non-linear programming to formulate and optimize data placement in the multi-cloud storage under complex requirements. We apply the euclidean distance measure through geometric space abstraction to construct the objective function, which helps get the final optimized data placement configurations. In this way, single objective as well as multi-objective optimization can be well addressed in Triones. Moreover, this model is general enough to adopt new factors and new requirements.
3) We have designed and implemented Triones. Experiments using Triones and the workloads collected from a file sharing application show that, compared to models in previous works, Triones is able to effectively satisfy both simple and complex requirements for the multi-cloud storage.

The remainder of this paper is organized as follows. In Section 2, we outline the background and motivation for Triones. Section 3 presents the systematic model in detail. The optimization method used in Triones is discussed in Section 4. In Section 5, we describe the design and implementation of Triones. We give the evaluation in Section 6 to show the effectiveness of Triones in satisfying different requirements on optimization. Related works are discussed in Section 7 and we conclude our paper in Section 8.

## 2 Background

### 2.1 Single Cloud Storage versus Multi-Cloud Storage

Single cloud storage relies on only one individual cloud storage provider. It is easy for developers to build their systems or applications over single cloud storage. What they need to do is only to use the APIs offered by the underlying cloud storage provider. This provider guarantees its service quality through its SLAs [20]. However, such SLAs will always be easily broken under the circumstance of using only one cloud storage provider. It is due to the fact that one provider cannot prevent transient failures that affect the service availability and permanent failures that cause permanent data loss from occurring [2], [3], [5]. For example, as reported on June 29, 2012, an outage caused by power issues made Amazon's services unavailable, which took down numerous companies such as Netflix and Pinterest [21]. Even worse, the accidents ever happened in Microsoft and Amazon resulted in permanent loss or corruption of users' data [3], [6]. Moreover, one has to face the issue of *vendor lock-in* in single cloud storage, that is, if he/she puts all data into one cloud storage provider, he/she is totally limited by this provider and it will be much expensive to switch from the provider to another [2].

To avoid the disadvantages of single cloud storage, most works [2], [4], [5], [6] have used multi-cloud storage to deploy the systems or applications. They stripe data into multiple cloud storage providers distributed in different geographical locations. This helps guarantee better fault tolerance, higher service availability as well as data durability, and avoiding vendor lock-in. In such cases, erasure coding is introduced to significantly reduce the amount of data to be stored compared to full replication [11], [12], [13], but with an overhead on read latency [6]. However, system or application developers can accept such overhead to achieve a lower cost for keeping a large amount of data. Such a scenario is common for a wide range of systems or applications providing services like data back-up, document archiving, or electronic health recording [2], [5], [6]. As a result, multi-cloud storage that is based on erasure coding is well suited to these types of systems or applications.

### 2.2 Optimization for Data Placement in the Multi-Cloud Storage

Many previous works used the multi-cloud storage just in ad-hoc ways to provide certain features. They only randomly chose the underlying cloud storage providers and parameter $(n, k)$ for their systems or applications. They did not further explore how far the optimization can be achieved in the multi-cloud storage by effectively choosing data placement configurations. A data placement configuration for the multi-cloud storage consists of $n$ cloud storage providers and parameter $(n, k)$ of erasure coding. Optimization is essential in this case as data placement configurations with different cloud storage providers and parameters $(n, k)$ offer totally different serving ability. Although Scalia [7] did adopt an adaptive scheme to choose data placement configurations at optimal cost, it only conducted single objective optimization and lacked a solution for multi-objective optimization.

Moreover, depending on the properties of the systems or applications, developers will hold various optimization requirements that are either simple or complex. Even within the same system or application, developers will have different requirements on data objects with diverse access patterns. Consequently, it is difficult for developers to cope with the data placement optimization issue themselves when deploying their systems or applications over the multi-cloud storage. A model that can formulate data placement optimization in a systematic way to address the issue is needed under this circumstance.

## 3 Triones—The Systematic Model

### 3.1 Formulation of Triones

The scenarios of how developers would deploy their systems or applications in the multi-cloud storage are shown in Fig. 1. The underlying cloud storage providers are located in different regions in the world. These providers only provide storage services and do not support running any program. Developers can use the multi-cloud storage in two modes. One is to build standalone services in independent datacenters to serve their users, but keep data in the multi-cloud storage. The other is to build service functions into end-user applications directly. Triones aims to target both scenarios.

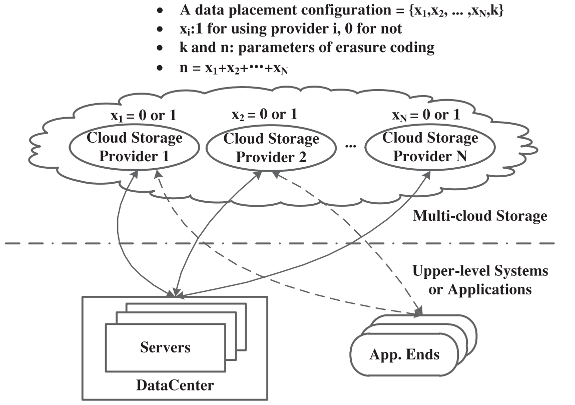


Fig. 1. The overview of how developers would build their systems or applications in the multi-cloud storage.

Linear Programming is limited to supporting different developer-required factors that may have complex definitions. We use non-linear programming [16], [17] for generally coping with data placement optimization. A lot of information should be taken into consideration while building the inequalities of constraints as well as the objective function of the non-linear programming. Each constraint inequality formula represents a constraint on one factor. The objective function is made up of factors that need to be optimized. The factor under consideration is built on top of basic variables and constants in Triones.

Basic variables are used to represent the results of optimization. They contain the information of cloud storage providers that are used for placing data. They also contain the information of erasure coding parameters. Thus, we can use a variable set, $X$, to denote these variables. $X$ takes the form:

$$X = \{x_1, x_2, \ldots, x_N, k\}. \tag{1}$$

$N$ is the total number of cloud storage providers. Each $x_i$ $(1 \leq i \leq N)$ equals to 1 or 0 that means the provider does appear or not in the final data placement configuration. The variable $k$ is the parameter of erasure coding. As the sum of $x_i$ is the value of another erasure coding parameter $n$, $n$ is not in the final result. All $x_i$ $(i = 1, 2, \ldots, N)$ and $k$ are the variables in the non-linear programming. A specific result of $X$, denoted as $X_s$ $(s = 1, 2, \ldots, S)$, is a data placement configuration. $S$ indicates the total number of data placement configurations that can be generated from $N$ cloud storage providers by using erasure coding. For example, assume there are four cloud storage providers, then one configuration may be $\{1, 1, 1, 1, k(3)\}$ while another configuration may be $\{1, 1, 0, 1, k(2)\}$. Such configurations could be used to place data for upper-level systems or applications.

The characteristics of underlying cloud storage providers can be regarded as coefficients for the non-linear programming. They are constants that are either provided by cloud storage providers completely through their SLAs on websites (e.g., guaranteed durability or pricing strategy) or collected by using test programs (e.g., access latency). These characteristics are the base to decide whether a data placement configuration can satisfy developers' requirements or not. Therefore, each cloud provider $i$ can map to a characteristic vector denoted as $\overrightarrow{P_i}$, and $\overrightarrow{P_i} = (C_{i,1}, C_{i,2}, \ldots, C_{i,K})$. In $\overrightarrow{P_i}$, each component reflects one characteristic for the $i$th cloud storage provider, with total $K$ characteristics under consideration. Thus, we can further have $P = [\overrightarrow{P_1}, \overrightarrow{P_2}, \ldots, \overrightarrow{P_N}]^{\mathrm{T}}$. $P$ is the characteristic matrix of all providers. It can be used as the constant coefficient matrix for constraint inequalities as well as the objective function of the non-linear programming.

Thus, the general form of the data placement optimization problem for multi-cloud storage systems can be expressed as follows:

$$\begin{cases} Minimize & : & f_1(X,P) \\ & : & f_2(X,P) \\ & : & \cdots \\ Subject\ to & : & f_1^{'}(X,P) \ < \ Con_1 \\ & : & f_2^{'}(X,P) \ < \ Con_2 \\ & : & \cdots \end{cases} \tag{2}$$

In Equation (2), $f_e$ $(e = 1, 2, \ldots)$ means a factor to be optimized in the objective function while $f_e^{'}$ $(e = 1, 2, \ldots)$ means a factor at the left-hand side of a constraint inequality. $f_e(X,P)$ and $f_e^{'}(X,P)$ indicate that all these factors are defined from the variable set $X$ and constant coefficient matrix $P$. $Con_e$ is the quantifiable constraint on $f_e$ in constraint inequalities. Notice that the factors in the constraint inequalities can also appear in the objective function of the non-linear programming. This is quite possible that, for example, developers might want a constraint on vendor lock-in level while still expecting to choose a result with the lowest vendor lock-in level. So far we have not defined the objective function. We will defer this until discussing the optimization. Developers can put multiple factors in the objective function to represent a complex requirement. The specific value of $X$ that could minimize the objective function is the data placement configuration that satisfies developers$\bar{a}$ complex requirements subject to constraints.

### 3.2 An Instance of Triones

For clarification, we consider five factors in Triones: vendor lock-in level, fault-tolerance level, access latency, cost, and availability. All these factors have been proved important for upper-level systems or applications by previous studies [2], [4], [5], [6], [7].

*Vendor lock-in level.* Vendor lock-in level is relevant with the number of cloud storage providers[2] ($n$, i.e., sum of $x_i$) in a data placement configuration. A larger $n$ means a less possibility that the data of upper-level systems or applications are limited by one or some particular providers. Hence, according to work [2], [7], we could use the reciprocal of $n$ to reflect vendor lock-in level in different configurations. That is, the vendor lock-in level can be formalized from the variables. The value of it is in range $(0, 1]$. Denote the vendor lock-in level as $V$, then it can be expressed as $V = 1/n$.

*Fault-tolerance level.* One of the most important benefits to place data in the multi-cloud storage is to tolerate transient or permanent failures of certain number of providers. We define fault-tolerance level, denoted as $F$, as $F = n - k$. Fault tolerance level is used to quantify the ability of a data placement configuration to tolerate failures. It is defined from variable set $X$. With erasure coding, a data placement configuration could tolerate $n - k$ cloud storage providers failing at the same time. Hence, the configuration with a higher value of $F$ can offer better availability and durability.

*Availability.* Only defining fault-tolerance level is not enough to reflect the actual reliability provided by a data placement configuration. For example, both parameters with $(n,k) = (3,2)$ and $(n,k) = (10,9)$ have the same fault-tolerance level, but the latter is less reliable because it is more likely to suffer from provider failures. Thus, we have to further define the availability and durability. The overall availability of a data placement configuration, denoted as $A$, can be defined as:

$$A = 1 - \sum_{m=n-k+1}^{n} \sum_{j=1}^{\binom{n}{m}} \prod_{i \in \mathbb{N}_j} (1 - a_i) \prod_{i^{'} \in \mathbb{N}_j^{'}} a_{i^{'}}. \tag{3}$$

Overall availability is based on the variable set $X$ as well as constant coefficient matrix $P$ (actually the availability column in the matrix). In Equation (3), $\mathbb{N}_j$ indicates a set of $m$ providers that are unavailable while $\mathbb{N}_j^{'}$ is a set of $n - m$ providers that still work normally. The $a_i$ $(a_{i^{'}})$ is the availability that provider $i$ $(i^{'})$ guarantees through its SLA, a constant appearing in $P$. The overall availability of a configuration is equal to the probability that no more than $n - k$ providers crash at the same time. This depends on the fact that events of going to crash are independent among these cloud storage providers [22], as one has no influence on others' service. The analysis of data durability is the same as availability, so we skip it for page limitation.

*Cost.* Cost is the factor that can also be derived from the variable vector $X$ and constant coefficient matrix $P$. In the multi-cloud storage, the cost can be defined as:

$$\begin{cases} C = \sum_{j=1}^{5} \sum_{i=1}^{T} \int x_i \times (Y_1 + Y_2) dt \\ Y_1 = \sum_{g=0}^{G_{i,j}-1} \left( D_{i,j,g} \times U_{i,j,g}(t) \right) \\ Y_2 = \left( Y_3 - \sum_{g=0}^{G_{i,j}-1} D_{i,j,g} \right) \times U_{i,j,G_{i,j}}(t) \\ Y_3 = \left( M_j \times R_j(t) + R_{i,j}^{'} \right) \\ \overrightarrow{M} = \left( \frac{1}{k}, \frac{1}{n}, \frac{1}{k}, \frac{k}{n}, 1 \right). \end{cases} \tag{4}$$

$C$ represents the cost to keep data objects in a data placement configuration. The $R_j(t)$'s $(j = 1, 2, \ldots, 5)$ are the resources consumed at time $t$: $R_1(t)$ is for the amount of storage, $R_2(t)$ is for the amount of transfer-out, $R_3(t)$ is for the amount of transfer-in, $R_4(t)$ is for the number of GET (Read) requests, and $R_5(t)$ is for the number of PUT (Write) requests. As one data object is divided into equal-sized blocks to be uploaded into the providers of a configuration, the real resources consumed on the data object is impacted by this configuration. $\overrightarrow{M}$ denotes the vector of impact parameters of a data placement configuration on cost. The $M_j$'s $(j = 1, 2, \ldots, 5)$ are for storage, transfer out, transfer in,

[2] Assume the providers work independently which means one will not influence others.

GET requests, and PUT requests. Moreover, the $R'_{i,j}$'s are sources that have been used in provider $i$.

Several cloud storage providers set ladders for charging. The price changes at different ladders. We take such pricing strategy into consideration for calculating cost. $U_{i,j,g}(t)$ represents the price of provider $i$ for data storage ($j=1$), data transfer-out ($j=2$), data transfer-in ($j=3$), GET requests ($j=4$), and PUT requests ($j=5$) at ladder $g$ and time $t$. If a provider does appear in a data placement configuration, its pricing strategy will be used for the calculation of cost. $D_{i,j,g}$ stands for the threshold of ladder $g$ in provider $i$ on $j$ (note that $D_{i,j,0}=0$). They are all characteristics of the providers in $P$. $G_{i,j}$ is the number of ladders used in provider $i$ on $j$.

*Access latency*. The access latency of a data object from site $r$ to a data placement configuration $X_s$, $L_{X_s,r}$, can be defined as follows:

$$L_{X_s,r} = \frac{1}{k} \times \max_{1 \le i \le N} \{x_i \times l_{i,r}\}. \tag{5}$$

In this equation, $l_{i,r}$ is the access latency of this data object from $r$ to cloud storage provider $i$. It is in the matrix $P$ and can be gotten by testing. Note that it includes latency of processing requests in the providers as well as transferring data through the Internet. Moreover, $l_{i,r}$ here refers to the latency tested under a fixed large size (e.g., 4 MB) instead of various sizes, the same as $L_{X_s,r}$. This is due to the fact that performance of accessing large-size data objects is enough to reflect the network conditions from a specific site $r$ to different cloud storage providers [2], [5], [6], [13]. In Triones, we use the access latency of 4 MB-size data objects as the criteria to measure a data placement configuration in providing access performance.

Recall that in data write, $n$ blocks of a data object have to be uploaded into $n$ providers. While in data read, $k$ providers of the $n$ ones would be chosen to retrieve $k$ blocks to reconstruct the original data object. As a result, the definition of access latency from site $r$ to a data placement configuration $X_s$ is based on the following assumption: the accessing actions can be executed in parallel [5]. As the amount of each block is $\frac{1}{k}$ of the original data object, $L_{X_s,r}$ can be roughly estimated as $\frac{1}{k}$ of the maximum value among all $l_{i,r}$, if provider $i$ is included in $X_s$. This definition is reasonable due to the fact that if one provider is quite slow, then the overall access latency will be increased.

Based on all the definitions, we can have a complex requirement of optimizing fault-tolerance level, access latency, and cost simultaneously. These three factors will be put in the objective function. We can also set three constraints on the optimization: a) vendor lock-in level should be no more than 0.2; b) fault-tolerance level should be no less than 1; c) guaranteed availability should be no less than 99.999 percent. The factors under constraint compose the inequality formulas.

## 4 Data Placement Optimization

### 4.1 The General Objective Function

One or more factors can be put in the objective function reflecting simple requirements or complex requirements. However, as discussed above, the factors considered have distinct definitions with each other. They have different units, differ significantly in the order of magnitude (e.g., cost and vendor lock-in level), and reflect independent aspects of a system. It is not easy to combine them reasonably under developers' complex requirements, not to mention new and unpredictable factors that may be raised in the future.

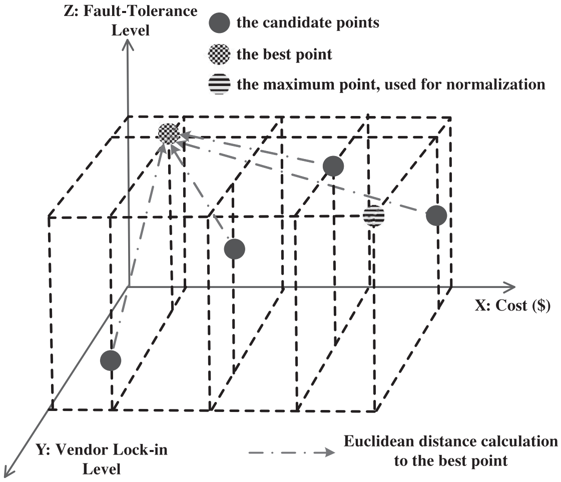


Fig. 2. A three-dimension geometric space for optimizing cost, fault-tolerance level, and vendor lock-in level.

Triones addresses this issue in the way through multi-dimension geometric space abstraction [18], [19]. In this geometric space, each dimension independently represents one factor. Based on the definition of these factors, every possible data placement configuration (i.e., $X_s$) can be used to calculate a specific value on each factor. Then each configuration maps to a point in this space, with the components of the point being the values of factors $F=\{f_1, f_2, \ldots, f_E\}$. $E$ is the total number of factors in the objective function. Configurations whose values of factors violate the constraints do not map to any point in the multi-dimension geometric space.

As illustrated in Fig. 2, for optimization, Triones sets a *best point* in the multi-dimension geometric space. The best point consists of the *best value* in each dimension. For example, in the dimension of access latency, the best value should be the lowest access latency to a specific data placement configuration. Notice that not all the best values are the smallest ones for corresponding dimensions. The *best* means the *smallest* for cost and vendor lock-in level while it means the *largest* for other factors such as availability and fault-tolerance level. It is decided by the definitions of the factors under consideration. Then, each point can get the distance to the best point by using euclidean distance measure. A point with a shorter distance to the best point will be considered as a better point and the corresponding configuration will be considered as a better schema.

However, the euclidean distance requires the same units of measurement for each dimension. Thus, we have to calculate the distance by using normalized values instead of absolute ones. The normalized value of a component in a point is calculated by its absolute value over maximum value in the corresponding dimension. The *maximum value* of every dimension composes a *maximum point*, which is also used for euclidean distance measure. The best point and the maximum point are usually virtual points that do

TABLE 1
Some Main Mathematical Notations Defined and Their Descriptions

| Definition | Description |
|---|---|
| $N$ | the number of cloud storage providers under consideration |
| $S$ | the number of data placement configurations generated from $N$ providers by using erasure coding |
| $P$ | the constant coefficient matrix, representing the characteristics of all providers, for the non-linear programming |
| $x_i$ | a boolean value that represents whether provider $i$ ($1 \leq i \leq N$) is used or not in the final configuration |
| $(n, k)$ | the erasure coding parameter ($1 < k < n = \sum_{i=1}^{N} x_i$) |
| $X$ | the optimization result consisting of $x_i$ and $(n, k)$ |
| $X_s$ | a specific value of $X$ that corresponds to a specific data placement configuration |
| $ED_s$ | The euclidean distance from the point corresponding to $X_s$ to the best point in the multi-dimension geometric space |

not match any data placement configuration. They are only meaningful for getting the optimized point (configuration).

With normalized values in all dimensions, Triones provides a general objective function in the form of euclidean-distance measure. Assume that there are $E$ factors to be optimized. Then in a $E$-dimension geometric space, a data placement configuration $X_s$ could map to a point $(f_1(X_s, P), \ldots, f_E(X_s, P))$. Moreover, the best point could be labeled as $(f_1^{best}, \ldots, f_E^{best})$ and the maximum point could be labeled as $(f_1^{max}, \ldots, f_E^{max})$. For the best and maximum points, $f_e^{best}$ $(e = 1, \ldots, E)$ is the best value among $f_e(X_s, P)$ $(1 \leq s \leq S)$ while $f_e^{max}$ $(e = 1, \ldots, E)$ is the maximum value among $f_e(X_s, P)$ $(1 \leq s \leq S)$. Then the euclidean distance from point of $X_s$ to the best point, denoted as $ED_s$, can be defined as:

$$ED_s = \sqrt{\sum_{e=1}^{E} W_e \times \left(\frac{f_e(X_s, P) - f_e^{best}}{f_e^{max}}\right)^2}. \tag{6}$$

$W_e$ is the optimization weight for factor $f_e$ and $\sum_{e=1}^{E} W_e = 1$. When developers require some factors to have more importance than others in the optimization, they can increase the optimization weight of these factors. To get a final result that optimizes (minimizes) this objective function, Triones traverses all possible data placement configurations. For each configuration, the value of each optimizing factor it could offer will be calculated. Then the configuration maps to a point in the multi-dimension geometric space. By setting a best point and a maximum point from the components of all points, each $ED_s$ can be gotten. Thus, the configuration with the smallest value of $ED_s$ corresponds to the optimized result of the objective function.

Note that this equation is also valid for simple requirements. Under this circumstance, the only factor derived from $X$ and $P$ is in Equation (6), the optimization weight for this factor is 100 percent, and the final result corresponds to the best point.

### 4.2 Discussion

Table 1 lists main mathematical notations defined and used in Triones. In addition, there are some features of Triones that need discussion:

*Generality.* As can be seen from the previous definitions, Triones is not oriented to any specific factor. It is general enough to adopt as many quantifiable factors as needed for data placement optimization in the multi-cloud storage. The combination of factors to be optimized can also be arbitrary. Thus, both single objective and multi-objective optimization can be conducted in Triones. This feature is quite useful for the long-time effectiveness of the model. New factors as well as new requirements on optimization will be raised quite possibly in the future. Even so, Triones can give system or application developers a powerful tool to get optimized results on these new factors and requirements, which cannot be realised by ways of choosing data placement configurations randomly.

*Dynamicity.* Triones does support dynamic parameters. The parameters can change in diverse ways, e.g., the joining of new cloud storage providers or the price variation of a provider. With a systematic definition, Triones is able to immediately update these parameter changes into variable set $X$ and constant coefficient matrix $P$. In this way, new variables and constants can be used to calculate new optimized configurations.

*Complexity.* Triones has to traverse all possible data placement configurations to find one that minimizes the objective function. The time complexity of this process is $O(2^N)$, in which $N$ is the number of cloud storage provider candidates. In the real world, $N$ is relatively small and usually less than 20 [6], [7]. Hence, the traverse process is still computationally feasible.

### 4.3 Data Migration

The changing of the optimized data placement configuration causes data migration. Several factors can cause configuration change such as the change of cloud storage provider parameters, optimization requirements, and data access patterns. However, the former two factors do not change often. For example, the eight cloud storage providers used in our evaluation all have their regular services for years. Rackspace Cloudfiles [29] keeps its pricing strategy unchanged for nearly two years. Amazon-S3 [20] only has two pricing changes in 2014. In terms of optimization requirements, developers will not change them frequently due to the characteristics of specific applications. As a result, the main trigger for data migration is the change of access patterns. Considerable access pattern variation can cause the frequent configuration change [7]. However, applications like data backup, document archiving and health-care recording, which dominate the usage of multi-cloud storage [2], [3], [4], [5], [6], [7], will suffer little data migration due to their low and steady access patterns.

With Triones, developers can get the optimized configurations for different access patterns. Besides, developers can also get the information on the cost, the real data size, and the real requests needed by a migration from Triones. We formulate these three factors as follows:

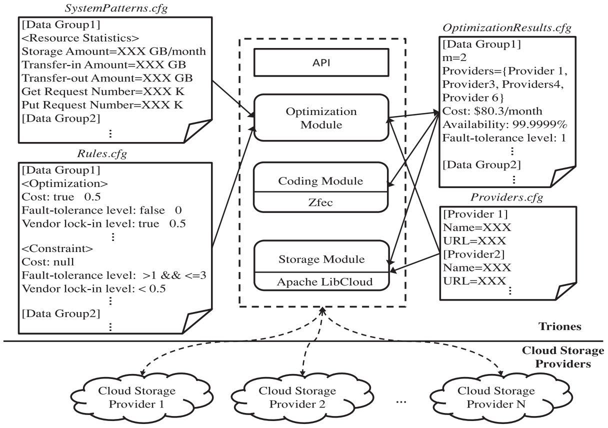
$$\begin{cases} O_c = \Omega_{out}(X_{old}, Q_s, Q_n) + \Omega_{in}(X_{new}, Q_s, Q_n), \\ Q_n^{'} = (k_{old} + n_{new}) \times Q_n, \\ Q_s^{'} = \left(1 + \frac{n_{new}}{k_{new}}\right) \times Q_s. \end{cases} \tag{7}$$

In this equation, $Q_n$ is the total number of the data objects needed for migration and $Q_s$ is their size. Migrating an object includes: reading the data blocks from the old configuration $X_{old}$ with the cost of $\Omega_{out}$, reconstructing the object, dividing the object with new configuration $X_{new}$, and finally uploading the new blocks with the cost of $\Omega_{in}$. Reconstructing the original data and re-encoding the data are performed locally, thus such cost is not considered. Equation (7) uses derived components of Equation (4), in which $\Omega_{out}$ is calculated by using $j = 2, 4$ ($Q_s = R_2(t), Q_n = R_4(t)$) and $\Omega_{in}$ is calculated by using $j = 3, 5$ ($Q_s = R_3(t)$, $Q_n = R_5(t)$). $O_c$ becomes large with the increasing of $Q_n$ and $Q_s$. Due to the properties of erasure coding, the real requests consumed ($Q_n^{'}$) during the migration and the real size of data needed to be transferred ($Q_s^{'}$) differ from $Q_n$ and $Q_s$ respectively, as shown in Equation (7).

The cost of data migration is orthogonal to the data placement configuration optimization studied in this paper. However, the cost is very important for developers to make decisions on whether to migrate or not, considering application specific characteristics. For example, if the optimized configuration change is temporal and the cost overhead is large, it will be wiser not to migrate. But if the benefits of the new optimized configuration cover the cost overhead and benefits of the old one, migration can be performed. As Triones does not have enough information of upper-layer systems or applications, automatic data migration is not possible. Thus, Triones provides migration information without making the final decision for developers.

## 5 Design and Implementation of Triones

In this section, we show the design and implementation of the model of Triones. It acts as a storage driver to help developers effectively place the data of their systems or applications in the multi-cloud storage. Fig. 3 presents the components that compose Triones. Triones has three process modules and four configuration files. These configuration files are either input for the modules or output of them. The three modules are responsible for calculating the optimization results, using erasure coding for data object dividing/reconstructing, and interacting with underlying cloud storage providers for transferring data blocks.



*SystemPatterns.cfg*

```
[Data Group1]
<Resource Statistics>
Storage Amount=XXX GB/month
Transfer-in Amount=XXX GB
Transfer-out Amount=XXX GB
Get Request Number=XXX K
Put Request Number=XXX K
[Data Group2]
        ⋮
```

*Rules.cfg*

```
[Data Group1]
<Optimization>
Cost: true   0.5
Fault-tolerance level: false   0
Vendor lock-in level: true   0.5
        ⋮
<Constraint>
Cost: null
Fault-tolerance level:  >1 && <=3
Vendor lock-in level: < 0.5
        ⋮
[Data Group2]
        ⋮
```

*OptimizationResults.cfg*

```
[Data Group1]
m=2
Providers={Provider 1,
Provider3, Providers4,
Provider 6}
Cost: $80.3/month
Availability: 99.9999%
Fault-tolerance level: 1
        ⋮
[Data Group2]
        ⋮
```

*Providers.cfg*

```
[Provider 1]
Name=XXX
URL=XXX
[Provider2]
Name=XXX
URL=XXX
        ⋮
```

Fig. 3. The components of Triones. They consist of three process modules and four configuration files.

We provide simple APIs for developers to set their requirements and operate data objects in the multi-cloud storage. Table 2 presents main APIs of Triones. Besides basic APIs like read, write, delete, and get the optimized configurations, Triones offers two explicit migration-relevant interfaces *getMigCost* and *getMigSize* to provide information on data migration.

### 5.1 Configuration Files

In the four configuration files, two files, i.e., the *Rules.cfg* and the *SystemPatterns.cfg*, are required to be set by developers manually. The other two files are managed by Triones automatically. In Triones, the optimization results are calculated for each data object that would be stored in the multi-cloud storage. Therefore, the requirements should also be set for each object. We use *data group* in Triones to classify data objects into groups and reduce the calculation overhead. A data group can contain only one data object or a set of data objects that have the same or very similar properties. Triones does not set strict rules on the classification of data objects. It is the developers who can decide how to batch different data objects into the same group. For example, one developer may choose to group objects of his/her system depending on their access times (10, 100, 1,000 s, etc.). Another developer would probably put all the objects into the same group, as he/she wants to get the same optimization result on vendor lock-in level for each object. As shown in Fig. 3, both developers' requirements and optimization results are based on the unit of data group.

*Rules.cfg*. The *Rules.cfg* is used for developers to set their requirements, i.e., optimization and constraint rules. In the *Optimization* section of each data group, each row represents a factor that is supported by Triones. The first column

TABLE 2
Main APIs Provided by Triones

| APIs | Description |
|---|---|
| $read(obj, conf, loc\_path)$ | read data object $obj$ from data placement configuration $conf$ into the local path $loc\_path$ |
| $write(obj, conf, loc\_path)$ | write data object $obj$ from the local path $loc\_path$ to data placement configuration $conf$ |
| $delete(obj, conf)$ | delete data object $obj$ within data placement configuration $conf$ |
| $getOptimizedConf()$ | calculate the optimized configurations for data placement in the multi-cloud storage |
| $getMigCost(objs, s, src\_conf, des\_conf)$ | get the cost of migrating $objs$ with total size $s$ from $src\_conf$ to $des\_conf$ |
| $getMigSize(objs, s, src\_conf, des\_conf)$ | get the real requests and data size of migrating $objs$ with total size $s$ from $src\_conf$ to $des\_conf$ |

*The objs is a list of data objects in a data group.*

following the factor name is a boolean value. A *true* indicates that this factor is demanded to be optimized for this group while a *false* means no optimization on it. The second column is a float value reflecting the optimization weight for this factor if it is set *true*. If developers only choose the factors to be optimized and do not set their optimization weights (i.e., the values in the second column of these factors are all 0), Triones will assign the same optimization weight to them.

**Algorithm 1.** Optimization Results Calculation

```
Input: $Rul.cfg$ for the $Rules.cfg$, $Sys.cfg$ for the $System Patterns.cfg$, $Pro.cfg$ for the $Providers.cfg$, $lastTime$, $success$
Output: $Opt.cfg$ for the $OptimizationResults.cfg$
 1: $global\ P[][]$                    $\triangleright$ $Constant\ coefficient\ matrix$
 2: $providers \leftarrow getProviders(Pro.cfg)$
 3: $lock()$
 4: if $currentTime() - lastTime > THRESHOLD$
      $or\ success == false$ then
 5:   $success \leftarrow false$
 6:   $lastTime \leftarrow currentTime()$
 7:   for $pro \in providers$ do
 8:     $update(pro.url, P)$
 9:   end for
10:   $flush(P)$
11:   $success \leftarrow true$
12: end if
13: $unlock()$
14: $data\_groups \leftarrow getDataGroups(Rul.cfg)$
15: $X \leftarrow getAllPlacementConfig(providers)$
16: for $dg \in data\_groups$ do
17:   $min \leftarrow MAXFLOAT$
18:   $res \leftarrow NULL$
19:   $csts \leftarrow getConstraints(Rul.cfg, dg)$
20:   $opts \leftarrow getOptReqAndWeight(Rul.cfg, dg)$
21:   $stts \leftarrow getStatistics(Sys.cfg, dg)$
22:   for $X\_s \in X$ do
23:     if $subtoConstraints(X\_s, csts, stts, P)$ then
24:       $insert(cands, X\_s)$
25:     end if
26:   end for
27:   $bp \leftarrow getBestPoint(cands, opts, stts, P)$
28:   $mp \leftarrow getMaximumPoint(cands, opts, stts, P)$
29:   for $X\_s \in cands$ do
30:     $ED\_s \leftarrow calED(X\_s, opts, stts, bp, mp, P)$
31:     if $ED_s < min$ then
32:       $min \leftarrow ED\_s$
33:       $res \leftarrow X\_s$
34:     end if
35:   end for
36:   $vals \leftarrow getValueofFactors(res, stts, P)$
37:   output $dg, res, vals$ to $Opt.cfg$
38:   clear $csts, opts, stts, cands$
39: end for
```

In the *Constraint* section of each data group, each row also stands for a factor that is supported by Triones. A rule specifying the constraints on the factor follows the factor name. The signs used in the constraint rule consist of $=$, $<$, $>$, $>=$, $<=$, &&, and ||. A *null* means no constraint is set on this factor. When Triones does incorporate new factors of cloud storage, developers only need to set the *Rules.cfg* if they intend to use them.

*SystemPatterns.cfg.* This configuration file is to present the system patterns of each data group. Currently, only resource statistics are needed. The *Resource Statistics* section of each data group specifies how the data objects in this group consume resources in upper-level systems. They are the total storage amount of these objects and their access times. With these information, the amount of transfer-in and transfer-out can also be calculated. These information can be easily offered by system or application developers as they can get or predict them through the system or application logs [7], [13], [23]. As developers may design their systems or applications in much diverse ways, we avoid directly analyzing logs of the systems or applications but require developers to provide such statistics [13]. These statistics are used to compute the cost of stripping these data objects in the multi-cloud storage, as shown in Equation (4). Note that to a data group, $R_j(t)^{'}s(j = 1, 2, \ldots, 5)$ are the resources consumed on the data objects of this group at time $t$.

*Providers.cfg.* The *Providers.cfg* contains the information (name and URL) of each candidate cloud storage provider used in Triones. An optimized data placement configuration is chosen depending on these candidate providers. The URLs are used by Triones to interact with these underlying cloud storage providers. When new cloud storage providers are joining in or old ones are removed, Triones will modify this configuration file and make these changes available.

*OptimizationResults.cfg.* After Triones successfully calculates the optimization results, it will output the result for each data group into the *OptimizationResults.cfg*. One result is a data placement configuration that satisfies developers' requirements subject to certain constraints. System or application developers then record these optimized configurations and use them for following data operations (like read, write, and migrate) over the multi-cloud storage. Besides, Triones also outputs the configurations' values on all factors (cost, availability, etc.) in the *OptimizationResults.cfg*, as displayed in Fig. 3. This helps developers get the benefits brought by the optimized configuration.

### 5.2 Process Modules

The process modules in Triones are the Optimization Module, the Coding Module, and the Storage Module.

*Optimization module.* The Optimization Module is responsible for choosing the optimized data placement configurations. When it is activated, it takes configuration files of the *Rules.cfg*, the *SystemPatterns.cfg*, and the *Providers.cfg* as input, and calculates optimization results into the *OptimizationResults.cfg*. The calculation procedure is summarized in Algorithm 1. As presented in Algorithm 1, the Optimization Module will firstly get the underlying cloud storage providers from the *Providers.cfg* (line 2) and update the constant coefficient matrix $P$ through the URLs of them (lines 7~9). As the characteristics of these providers (such as price strategies and guaranteed availability) do not change so often, we set THRESHOLD to 1,209,600 seconds (two weeks) to ensure that the characteristics can be updated every two week (lines 4). In Triones, the constant coefficient matrix is kept globally (line 1). Hence, synchronization is needed to

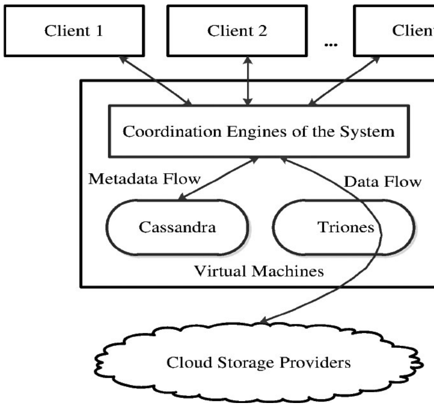


Fig. 4. The architecture of the prototype storage system that uses Triones to store data in the multi-cloud storage.

TABLE 3
The Information of Eight Cloud Storage Providers

| Provider | Price (at the first ladder) | | | | Location |
|---|---|---|---|---|---|
| | Storage | Transfer out | GET Req. | PUT Req. | |
| GS | 0.026 | 0.12 | 0.01 | 0.1 | Default |
| S3-IRL | 0.03 | 0.12 | 0.004 | 0.05 | Ireland |
| S3-TKY | 0.033 | 0.201 | 0.0037 | 0.047 | Japan |
| S3-CA | 0.033 | 0.12 | 0.0044 | 0.055 | USA |
| S3-SA | 0.0408 | 0.25 | 0.0056 | 0.07 | Brazil |
| CF-SYD | 0.1 | 0.12 | 0 | 0 | Australia |
| CF-VA | 0.1 | 0.12 | 0 | 0 | USA |
| CF-HKG | 0.1 | 0.12 | 0 | 0 | HongKong, China |

*The prices for data storage, data transfer out, and GET/PUT requests are measured in dollars/month*GB, dollars/GB, and dollars/10,000 requests respectively. GS is short for Google Storage and CF is short for Cloudfiles.*

avoid uncorrect optimization calculation when $P$ is being updated (line 3 and line 13). But the overhead of synchronization can be neglected, as the updating action occurs only every two week. After the updating action completes, the constant coefficient matrix will be flushed to disks for permanence (line 10). We also use a boolean flag *success* to indicate whether the updating and flushing actions complete successfully. If failures occur in Triones during the execution of any action, they will be re-executed after Triones restarts as *success* is kept *false* (lines 4~5). The modification of *lastTime* and *success* can be made permanent by logging [24].

The Optimization Module then gets the data groups from the *Rules.cfg* (line 14) as well as all possible data placement configurations (line 15). To each data group, the Optimization Module firstly obtains the constraint rules for it (line 19), and omits the data placement configurations that cannot be mapped into the multi-dimension geometric space (lines 22~26). In the next step, the euclidean distance associated with each candidate data placement configuration is calculated (lines 29~ 35) through the best point and the maximum point (lines 27~28). The data placement configuration with the minimum euclidean distance is the final optimization result for this data group (lines 32~33). Such configuration, along with its values on all factors, will be output to the *OptimizationResults.cfg* for developers (lines 36~37) in human-readable format, as displayed in Fig. 3.

*Coding module.* Erasure coding is applied in the Coding Module for data blocks encoding and decoding. When a write request of a data object arrives, the Coding Module divides this object into $k$ data blocks and generate another $n-k$ encoded blocks. These $n$ data blocks will be uploaded to the $n$ underlying cloud storage providers in the data placement configuration provided by developers. The real data transfer will be performed in the Storage Module. In reading a data object, $k$ data blocks will be retrieved for the Coding Module to reconstruct the original object.

*Storage module.* The Storage Module is in charge of interacting with underlying cloud storage providers to transfer (PUT or GET) data blocks. A main function of the Storage Module is to mask the differences among these cloud storage providers. In uploading $n$ blocks of a data object, the Storage Module returns only when all the $n$ underlying cloud storage providers reply successfully. This is to ensure the fault-tolerance level required by system or application developers. In retrieving $k$ data blocks, the Storage Module only select $k$ providers instead of all the $n$ ones. We use a principle for the selection: if access latency is required to be optimized, the $k$ fastest providers are selected, or the $k$ cheapest ones are selected. In the case that the Storage Module fails to retrieve blocks from one or more providers, the $(k+1)th, (k+2)th, \ldots,$ fastest or cheapest providers will be selected. In addition, the Storage Module will create multiple threads to communicate with these cloud storage providers in data read and write. This helps to make the most of parallelism provided by the underlying providers and improve the access performance.

Both the Coding Module and the Storage Module are stateless and do not need centralized coordination. Thus, they scale well for developers to build their systems or applications.

### 5.3 Implementation

We implement Triones in Python in 2,000 LOCs. The APIs exposed to developers are also in Python. In the Coding Module, we use Zfec [25], an open-source library that adopts fast classic RS-coding [8], to implement erasure coding. In addition, we use Apache LibCloud [26] in the implementation of the Storage Module. With Apache LibCloud, new cloud storage providers can be easily added to and supported by the Storage Module.

## 6 Evaluation

### 6.1 Experimental Setup

We have deployed a prototype storage system to evaluate the effectiveness of Triones. It has run in experimental virtual machines built in Singapore through VPS [27]. As illustrated in Fig. 4, the storage system keeps metadata in Cassandra in the local, and stores its data objects in the multi-cloud storage through Triones. The metadata contains the data placement optimization results. The design details of this prototype system is beyond the scope of this paper, so we do not discuss them further. The evaluation is only conducted for Triones instead of the system.

We choose eight commercial cloud storage providers [20], [28], [29] as the candidate storage backends for data placement. They are listed in Table 3. We consider cost,

TABLE 4
Experimental Data Sets

| | Data Objects | Storage | Transfer Out | GET Num. | Put Num. |
|---|---|---|---|---|---|
| Set 1 | 9,930 | 9.46 GB | 9,465.39 GB | 9,929.7 k | 14.86 k |
| Set 2 | 99,905 | 292.36 GB | 732.25 GB | 250.12 k | 0 |

access latency, fault-tolerance level, vendor lock-in level, durability, and availability for data placement configurations in Triones. These factors have been defined in Section 3. As all these eight cloud storage providers cost free for data transfer-in, we do not consider this for calculating the cost. The guaranteed durability and availability of all the providers are 99.9999 and 99.9 percent respectively.

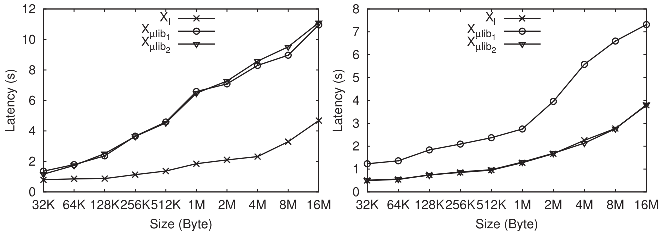
We simulate the behaviors of the storage system according to the workloads collected from a real file sharing application (http://thu.meepo.org/) used in Tsinghua University. The workloads captured the access logs from many users. As shown in Table 4, two data sets containing different number of data objects with different access patterns are collected. Data Set 1 consists of data objects with relatively high access frequency. Each data object was accessed for around 1,000 times in a month, most of which were read requests. Data Set 2 provides cold data as each object was read for less than five times in a month, with no write requests. Each data set maps to a data group in Triones.

We set unified constraints on each data group to Triones. Vendor lock-in level is no larger than 0.5. This means at least two providers should be included in the final optimized configurations. Durability of the constructed configuration is no less than 99.999999 percent while availability is no less than 99.999 percent. Moreover, it should tolerate failures of at least one provider, i.e., the fault-tolerance level is no less than 1. Note that for readability, we do a little change of Equation (1) to represent the final data placement configuration in the evaluation. We use the name of the cloud storage provider instead of the number 1 if this provider (i.e., $x_i$) is chosen in the configuration. If one provider is not chosen, its name as well as the number 0 does not occur in the final configuration.

## 6.2 Single Objective Optimization

This section provides the results for optimization on a single objective. Actually there are many single factors that can be optimized. Due to space limitation, we only report the results of cost and access latency here. These two factors are cared about more than others by developers when deploying systems or applications in the multi-cloud storage [2], [3], [4], [7]. To test access latency, we set a *measurement unit* that consists of a write request followed by a read request [5], [6]. Each unit is executed periodically (every one minute) in the prototype storage system. To each request, the size of data object is chosen randomly.

*Cost.* For Data Set 1, Triones calculates an optimized data placement configuration, $X_c^1$,[3] to keep the data objects. $X_c^1$ uses {CF-SYD, CF-HKG, GS, $k(2)$}. Among all the configurations that can be built from the eight cloud storage providers, $X_c^1$ offers the lowest cost,[4] \$1122.37, for Data Set 1. For Data Set 2, Triones applies {S3-IRL, S3-CA, GS, $k(2)$} to build another configuration $X_c^2$. The cost of $X_c^2$ is \$101.19, which is still the lowest to keep data objects of Data Set 2 that are mainly for storage. As data objects in Data Set 1 are frequently accessed, the cloud storage providers chosen are relatively cheap in access. On the other hand, the providers in $X_c^2$ are cheap in data storage. If we use $X_c^1$ for Data Set 2, the cost will be \$121.07. This is about 19.65 percent higher than that of $X_c^2$. Similarly, if we move data objects of Data Set 1 into $X_c^2$, the cost rises to \$1127.44. Note that for Data Set 1, GS uses two pricing ladders on data transfer-out while other providers only use one. For Data Set 2, all providers only use one pricing ladder.



(a)Write (b)Read

Fig. 5. Access latency from the virtual machines in Singapore to $X_l$, $X_{\mu lib_1}$, $X_{\mu lib_2}$.

We further show how Triones works when more pricing ladders are involved by simulation. We extend Data Set 1 to change its transfer-out size from 1 to 400 TB (keeping each data object with 1,000 get requests). Meanwhile, we vary the storage of Data Set 2 from 1 to 400 TB (keeping each data object with 1 get request). Triones adapts well to more pricing ladders and generates the corresponding results to achieve optimal cost. For the changing of Data Set 1, Triones uses $X_{ladder_1}^1$:{CF-SYD, CF-HKG, CF-VA, $k(2)$} if the transfer-out amount is less than 6 TB, and $X_{ladder_2}^1$:{GS, CF-HKG, CF-VA, $k(2)$} if the amount is larger than 6 TB. When the transfer-out is 400 TB (four ladders are reached on transfer-out), using $X_{ladder_1}^1$ costs \$33,282.4, higher than that (\$33,184.19) of $X_{ladder_2}^1$. However, when the transfer-out is only 4 TB, $X_{ladder_2}^1$ brings higher cost (\$483.98) than $X_{ladder_1}^1$ (\$480.6). For Data Set 2, Triones generates the same result as $X_{ladder}^2$:{GS, S3-IRL, S3-CA, $k(2)$} despite the number of pricing ladders involved.

*Access latency.* For access latency optimization, Triones correspondingly chooses a data placement configuration $X_l$. $X_l$ utilizes {S3-TKY, GS, CF-SYD, CF-HKG, $k(3)$}. The work of $\mu$LibCloud [6] shows that a data placement configuration with $n = 5$ and $k = 3$ could reduce data access latency, especially read latency. It randomly uses two configurations as $X_{\mu lib_1}$:{S3-IRL, S3-CA, S3-SA, GS, CF-VA, $k(3)$} and $X_{\mu lib_2}$:{S3-TKY, S3-CA, S3-SA, CF-SYD, CF-HKG, $k(3)$}.

We compare the optimized configuration computed by Triones to the ones used in $\mu$LibCloud. As presented in Fig. 5, $X_l$ outperforms both $X_{\mu lib_1}$ and $X_{\mu lib_2}$ in data

[3] The superscript numbers of $X$ refer to the data sets. The subscript letters of $X$ refer to the factors to be optimized, in which $c$ is for cost, $l$ is for access latency, $f$ is for fault-tolerance level, and $v$ is for vendor lock-in level.

[4] The cost discussed in the evaluation refers to the dollars one has to pay in a month.

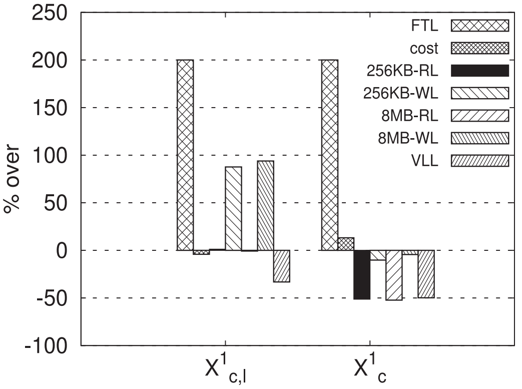


Fig. 6. The cost, fault-tolerance level, read latency, write latency (WL), and vendor lock-in level (VLL) of $X^1_{c,l,v,f}$ in comparison with $X^1_{c,l}$ and $X^1_c$.

access latency[5] from Singapore to the multi-cloud storage. For example, in writing 16 MB-size objects, the latency to $X_l$ is 4.68 s. It is about 57.26 and 57.84 percent lower than that to $X_{\mu lib_1}$ and $X_{\mu lib_2}$ respectively. The latency of reading 16 MB-size objects from $X_l$ is 3.78 s, about 48.22 percent less than that from $X_{\mu lib_1}$ and similar to that from $X_{\mu lib_2}$.

As mentioned in Section 5, Triones chooses $k$ fastest providers in a configuration to retrieve data blocks for data read, if access latency is one of the optimization objectives. Both of $X_l$ and $X_{\mu lib_2}$ have $k = 3$ as well as cloud storage providers including S3-TKY, CF-SYD, and CF-HKG. These three providers are the fastest three ones among the eight to the virtual machines located in Singapore. Hence, we can see from Fig. 5b that read latency from $X_l$ and $X_{\mu lib_2}$ differ little from each other. Besides, in data write, Triones must put all $n$ data blocks to $n$ providers composing of a data placement configuration, no matter how slow one provider is. That is why write latency is higher than read latency for a configuration, as shown in Fig. 5a.

### 6.3 Multi-Objective Optimization

For multi-objective optimization in Triones, there can be numerous combinations of factors to be optimized. We have chosen the optimization requirements depending on the features of these two data sets to show how Triones works on multi-objective data placement optimization. We firstly set the objectives to have the same optimization weight.

*Data Set 1.* For Data Set 1 with intensive access, we prefer both cost and access latency to be optimized at the same time. Triones uses a data placement configuration, $X^1_{c,l}$, to satisfy the complex optimization requirement. $X^1_{c,l}$ happens to be the same as $X_l$, which is the configuration for only optimizing access latency. The cost of $X^1_{c,l}$ is \$1,325.33, 18.08 percent more expensive than $X^1_c$. However, for both read and write operations, $X^1_{c,l}$ could reduce the latency on different sizes (from 32 KB to 16 MB) by 30∼50 percent compared to $X^1_c$. This is reasonable when cost is not the only concern, i.e., cost and access latency have the same optimization weight. Compared to $X^1_{c,l}$, the cost of $X_{\mu lib_1}$ and $X_{\mu lib_2}$ are \$1,388.69 and \$1,544.67, which is 4.78 and 16.55 percent higher respectively. Thus, we can see that a randomly chosen data placement configuration can hardly satisfy complex requirements involving optimizing multiple factors in the multi-cloud storage. This further proves the necessity of a systematic model for both single and multiple objective optimization.

Now suppose we want to further consider optimizing fault-tolerance level and vendor lock-in level for Data Set 1. A higher fault-tolerance level could help improve the availability of access services. Meanwhile, a lower vendor lock-in level helps mitigate the limitation of the underlying cloud storage providers. The requirement is to optimize cost, access latency, fault-tolerance level, and vendor lock-in level.

In this case, Triones computes the data placement configuration $X^1_{c,l,v,f}$ as {S3-TKY, S3-IRL, S3-CA, GS, CF-SYD, CF-HKG, $k(3)$}. It is effective in balancing more factors. Compared to $X^1_c$, $X^1_{c,l,v,f}$ improves fault-tolerance level by 200 percent, reduces vendor lock-in level by 49.85 percent, and reduces read latency (RL) by about 50 percent, as presented in Fig. 6. However, the cost of $X^1_{c,l,v,f}$ rises by 13.19 percent to keep more redundant data blocks. In contrast with $X^1_{c,l}$, $X^1_{c,l,v,f}$ reduces cost by 4.14 percent as well as achieving 3× fault-tolerance level. Moreover, the vendor lock-in level of $X^1_{c,l,v,f}$ is reduced by 33.2 percent. This is at the expense of access latency. For example, $X^1_{c,l,v,f}$ brings 93.92 percent higher latency in writing 8MB-size data objects than $X^1_{c,l}$. But in data read latency, $X^1_{c,l,v,f}$ and $X^1_{c,l}$ differ little from each other because of the reading mode used in Triones, as discussed in Section 6.2.

TABLE 5
The Cost, Vendor Lock-in Level, and Fault-Tolerance Level of $X^2_{c,v,f}$ and $X^2_c$

| | Cost (\$) | Vendor Lock-in Level | Fault-tolerance level |
|---|---|---|---|
| $X^2_c$ | 101.19 | 0.333 | 1 |
| $X^2_{c,v,f}$ | 175.01 | 0.125 | 6 |

*Data Set 2.* Data Set 2 contains data objects that are mainly stored with little access. To such access patterns, we are interested in cost, vendor lock-in level, and fault-tolerance level for them in the prototype storage system. Fault-tolerance level is essential to guarantee the durability of cold data that will be stored in the cloud storage for a long time. We do not consider optimizing access latency because of the low access frequency of Data Set 2. The complex requirement here is the optimization for all these three factors. Triones gets a data placement configuration, $X^2_{c,v,f}$, which uses all the eight providers and $k = 2$. As listed in Table 5, $X^2_{c,v,f}$ performs well in cost, vendor lock-in level, and fault-tolerance level at the same time. Compared to $X^2_c$, $X^2_{c,v,f}$ reduces the vendor lock-in level by 62.46 percent. Besides, the fault-tolerance level of $X^2_{c,v,f}$ is improved by 5×. However, as more data blocks have to be kept, the cost of $X^2_{c,v,f}$ increases by 72.95 percent compared with that of $X^2_c$.

[5] The access latency discussed in the evaluation refers to 90th-percentile access latency. Moreover, the access latency does not contain the time for erasure coding. The overhead of data encoding/decoding is discussed in Section 6.4.

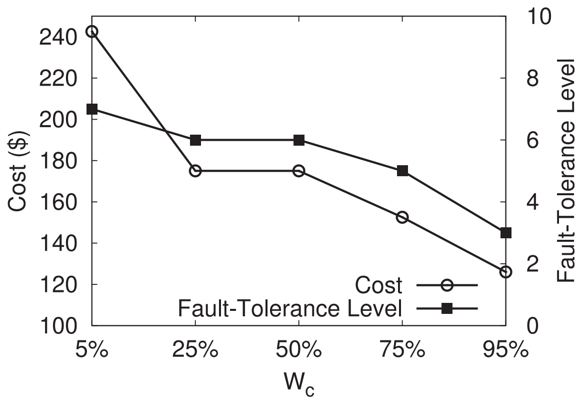


Fig. 7. The cost and fault-tolerance level of data placement configurations under different $W_c$.

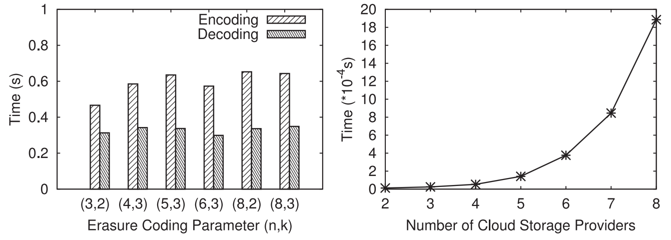
*Varying optimization weight.* Now let's see how Triones balances among multiple objectives in different optimization weight. We consider optimizing cost and fault-tolerance level simultaneously for data objects in Data Set 2. Thus, we use $W_c$ to represent the optimization weight for cost and $W_f$ for fault-tolerance level, where $W_c + W_f = 1$. Fig. 7 presents the cost and fault-tolerance level of configurations that are calculated under different value of $W_c$. We can observe that Triones is able to choose data placement configurations to balance among cost and fault-tolerance level, according to their optimization weight. For example, when we set $W_c = 5\%$ and $W_f = 95\%$, it means that we place fault-tolerance level at a more important position in the optimization. Triones gets a configuration that provides cost of $242.57 and fault-tolerance level of 7 (k=1 is allowed in this case). While we set $W_c = 95\%$ and $W_f = 5\%$, which means cost is more cared about, a configuration with cost of $126.01 and fault-tolerance level of 3 is used. In this configuration, the cost decreases by 48.05 percent (more important) while the fault-tolerance level decreases by 57.14 percent (less important), compared to that of $W_c = 5\%$.

## 6.4 Overhead of Triones

Triones is based on erasure coding and non-linear programming. Hence, the overhead of the model consists of the time to encode/decode data blocks as well as computing an optimization result.

Fig. 8a shows the time spent in encoding/decoding blocks of 16 MB-size data objects under different erasure coding parameters of $(n, k)$. These parameters are chosen according to the configurations built above. We still use Zfec for the experiment. The time spent under each parameter is averaged over 10,000 tests. We can see that the time spent in decoding is about 50 percent less than that in encoding. Even so, both of them are about 0.5 s in 16 MB-size data objects. This overhead is a small part of the total latency of accessing these data objects in the multi-cloud storage. We think the overhead imposed by erasure coding is reasonable given the benefits it brings to stripe data in the multi-cloud storage, compared to single cloud storage (e.g., availability or vendor lock-in) or full-replication policy (e.g., cost).

As discussed in Section 4 and Section 5, the complexity of Algorithm 1 is $O(2^N)$. Fig. 8b illustrates that when the number of cloud storage providers increases, the time for the calculation of Triones on one data group rises exponentially. But as $N$ is usually small, the complexity is not a concern here. For example, when $N$ is 8, the overhead of getting an optimized configuration on one data group is only 0.0018 s. However, with new cloud storage providers are made available, this calculation might take longer time. Fortunately, the distance measure can be performed independently for each configuration, i.e., the calculation can be parallelized [30] to reduce the time. Furthermore, the Pruning Algorithms [31] can be used to further reduce the complexity of the calculation.



(a) Erasure Coding (b) Optimization Calculation

Fig. 8. The overhead of Triones. The results are tested in the machine with Intel Xeon X5650 (2.67 GHz) and 16 GB RAM.

## 6.5 Configuration Change Frequency

We have used a trace on 100 documents (each around 3 MB) to test the frequency of configuration change. This trace is similar to that used in Scalia [7] and was collected from the file sharing application in Tsinghua University. For the first ten days after being uploaded, each document got 150~200 reads per day. The access frequency decreased from 150 times/day to 1 time/day in the next twenty days. Then the frequency kept steady (below three times/day) ever after.

Four optimization requirements are set: optimizing cost (denoted as $OR_1$), optimizing latency ($OR_2$), optimizing both cost and access latency ($OR_3$), and optimizing cost, access latency, fault-tolerance level as well as vendor lock-in level ($OR_4$). For the complex requirements, every objective has the same weight.

As for the results, three configuration changes happen for $OR_1$. $OR_2$, $OR_3$, and $OR_4$ do not incur any configuration change. For $OR_1$, configuration changes occur at the access frequencies of 12 times/day, 10 times/day, and five times/day respectively. This is because when access frequency decreases, the cost of data storage accounts for a larger proportion and causes the change of the data placement configuration intending cheaper cost on storage. It seems that the frequencies causing the configuration change are close to each other. However, this is intrinsic for the optimization based on the provided information and will not influence the effectiveness of Triones for $OR_1$. In fact, the optimized configuration for one frequency is suboptimal for anther. For example, when the access frequency decreases to five times/day, if the configuration ({S3-IRL, S3-CA, CF-VA, $k(2)$}) generated for 10 times/day is still used, the cost is $0.176857. This is just slightly higher than the cost ($0.176829) of the optimized configuration ({S3-IRL, S3-CA, GS, $k(2)$}) for 5 times/day. The migration costs of the documents between these two configurations (in two different directions) are \$0.03633 and \$0.03726 respectively. As a result, if the access frequency remains in the range of 10 times/day and five times/day (or in the around ranges), keeping the configurations without data migration might be a good decision. Certainly, developers can rely on Triones to make other migration decisions. In any case, Triones can work effectively for $OR_1$ compared to randomly choosing data placement configurations. On the other hand, $OR_2$ does not cause the change of configuration because $OR_2$ only optimizes latency while access pattern change only influences cost. For $OR_3$ and $OR_4$, the cost changed by access pattern has less impact on changing the configuration for optimizing multiple objectives.

## 7 Related Work

Many previous works are related to Triones and have inspired Triones. We review these works in this section.

*Multi-cloud storage based on erasure coding.* RACS [2] was the first work to put forward the idea of applying erasure coding in the multi-cloud storage. The goal was to avoid vendor lock-in and improve the service availability. HAIL [4] aimed at protecting the availability and security of data by using erasure coding in the multi-cloud storage. DepSky [5] tried to guarantee the reliability and security of applications through encryption and encoding at reasonable cost and access latency. $\mu$LibCloud [6] attempted to achieve uniform access and reduce access latency under such circumstance. However, these studies concentrated on using the multi-cloud storage in ad-hoc ways. The schemas or models in their systems only randomly chose data placement configurations to achieve certain features. Compared with them, Triones is a systematic model to address the optimization issue for developers in the multi-clouds storage. It enables them to deploy their systems or applications in an optimized way under their simple or complex requirements. One similar work to Triones is Scalia [7]. Scalia used an adaptive scheme to choose different data placement configurations for offering the optimal cost while satisfying certain constraints. However, from Triones' point of view, the model in Scalia only conducted single objective optimization. Triones does the work for both single objective as well as multi-objective optimization.

*Optimization in cloud platforms.* Optimization has been studied in cloud platforms by previous studies for years. For example, the work in [32] focused on multi-objective optimization for deploying online social services in master-slave cloud platforms. The work in [33], [34] proposed optimization solutions for the scheduling of many dynamic computing tasks to satisfy multiple objectives. Compared to these studies, the assumptions and usage scenarios of multi-cloud storage are different. As mentioned in Section 3, the providers in the multi-cloud storage do not support executing any code [2], [5], [7]. They can only be used by their storage interfaces without modifications. How tasks are running and managed in these cloud storage providers cannot be gotten outside. Moreover, in multi-cloud storage that is based on erasure coding, the factors (e.g., vendor lock-in or availability) cared about by system or application developers have specific characteristics. Thus, optimization in such scenario has to be explored.

*Coding schemes.* The design of different coding schemes have been widely studied in storage systems. NCCloud [3] proposed to use functional minimum-storage regenerating code to reduce the cost of storage repair in the multi-cloud storage if one cloud storage provider fails permanently. The work in [35] presented how to use the least amount of data for XOR-based erasure coding during recovery and designed a new coding scheme based on RS-code to deal with degraded reads. In the in-house storage scenario, Windows Azure employed Local Reconstruction Codes (LRC) [36] to reduce the number of data blocks that have to be read to reconstruct lost or corrupted blocks, without improving the storage overhead. Triones does not do any improvement in coding schemes. We use the classic RS-code [8] to stripe data in the multi-cloud storage. Our work is orthogonal to this category of related works. New coding schemes can be applied in the Coding Module and Triones can then conduct new optimization with these coding schemes.

## 8 Conclusion

This paper presents Triones, a systematic model to formulate and optimize data placement in multi-cloud storage by using erasure coding. As a systematic approach, Triones tries its best to avoid ad-hoc ways of randomly choosing data placement configurations. It uses non-linear programming to define the problem of data placement optimization. In this model, quantifiable factors under consideration can be expressed in the inequalities of constraints as well as being put in the objective function. We apply euclidean distance measure through geometric space abstraction for the objective function to calculate the optimization results. In this way, complex requirements that are not considered in previous works can be easily included in Triones. Furthermore, new factors and requirements can be adopted in the model and optimized by the same means. Triones helps system or application developers to achieve the features of the multi-cloud storage in an optimized way with reasonable overhead.

## Acknowledgments

The authors would like to thank the anonymous reviewers for their constructive comments. This work is supported by National High-Tech R&D (863) Program of China (2013AA01A213), Natural Science Foundation of China (61433008, 61373145, 61170210, U1435216), Chinese Special Project of Science and Technology (2013zx01039-002-002).

## References

[1] Computer sciences corp [Online]. Available: http://www.csc.com/, Aug. 2015.

[2] 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.

[3] H. Chen, Y. Hu, P. Lee, and Y. Tang, "NCCloud: A network-coding-based storage system in a cloud-of-clouds," *IEEE Trans. Comput.*, vol. 63, no. 1, pp. 31–44, Jan. 2014.

[4] K. D. Bowers, A. Juels, and A. Oprea, "Hail: A high-availability and integrity layer for cloud storage," in *Proc. 16th ACM Conf. Comput. Commun. Security*, 2009, pp. 187–198.

[5] A. Bessani, M. Correia, B. Quaresma, F. André, and P. Sousa, "DepSky: Dependable and secure storage in a cloud-of-clouds," *ACM Trans. Storage*, vol. 9, no. 4, p. 12, 2013.

[6] S. Mu, K. Chen, P. Gao, F. Ye, Y. Wu, and W. Zheng, "$\mu$libcloud: Providing high available and uniform accessing to multiple cloud storages," in *Proc.ACM/IEEE 13th Int. Conf. Grid Comput.*, 2012, pp. 201–208.

[7] 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.

[8] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *J. Soc. Ind. Appl. Math.*, vol. 8, no. 2, pp. 300–304, 1960.

[9] R. C. Singleton, "Maximum distance $q$-nary codes," *IEEE Trans. Inf. Theory*, vol. IT-10, no. 2, pp. 116–118, Apr. 1964.

[10] V. Pless, *Introduction to the Theory of Error-Correcting Codes*, vol. 48. New York, NY, USA: Wiley, 2011.

[11] H. Weatherspoon and J. D. Kubiatowicz, "Erasure coding vs. replication: A quantitative comparison," in *Proc. Revised Papers 1st Int. Workshop Peer-to-Peer Syst.*, 2002, pp. 328–338.

[12] 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.

[13] Z. Wu, M. Butkiewicz, D. Perkins, E. Katz-Bassett, and H. V. Madhyastha, "Spanstore: Cost-effective geo-replicated storage spanning multiple cloud services," in *Proc. 24th ACM Symp. Operating Syst. Principles*, 2013, pp. 292–308.

[14] W. S. Shin and A. Ravindran, "Interactive multiple objective optimization: Survey I – continuous case," *Comput. Operations Res.*, vol. 18, no. 1, pp. 97–114, 1991.

[15] R. T. Marler and J. S. Arora, "Survey of multi-objective optimization methods for engineering," *Struct. Multidisciplinary Optimization*, vol. 26, no. 6, pp. 369–395, 2004.

[16] D. P. Bertsekas, *Nonlinear Programming*. Belmont, MA, USA: Athena Scientific, 1999.

[17] H. W. Kuhn, "Nonlinear programming: A historical view," in *Traces and Emergence of Nonlinear Programming*. New York, NY, USA: Springer, 2014, pp. 393–414.

[18] J. A. Nelder and R. Mead, "A simplex method for function minimization," *Comput. J.*, vol. 7, no. 4, pp. 308–313, 1965.

[19] P. Hinker and C. Hansen, "Geometric optimization," in *Proc. 4th Conf. Vis.*, 1993, pp. 189–195.

[20] Amazon s3 [Online]. Available: http://aws.amazon.com/s3/, Aug. 2015.

[21] Amazon outage [Online]. Available: http://venturebeat.com/2012/06/29/amazon-outage-netflix-instagram-pinterest/, Jun. 2012.

[22] 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. 9th USENIX Symp. Operating Syst. Des. Implementation*, 2010, pp. 61–74.

[23] P. Bodik, A. Fox, M. J. Franklin, M. I. Jordan, and D. A. Patterson, "Characterizing, modeling, and generating workload spikes for stateful services," in *Proc. 1st ACM Symp. Cloud Comput.*, 2010, pp. 241–252.

[24] V. Prabhakaran, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Analysis and evolution of journaling file systems." in *Proc. USENIX Annu. Techni. Conf., General Track*, 2005, pp. 105–120.

[25] Python-zfec [Online]. Available: https://pypi.python.org/pypi/zfec/, Aug. 2015.

[26] Apache libcloud [Online]. Available: http://libcloud.apache.org, Aug. 2015.

[27] Cloud vps server hosting [Online]. Available: http://vps.net, Aug. 2015.

[28] Google cloud storage [Online]. Available: https://cloud.google.com/storage/, Aug. 2015.

[29] Rackspace cloudfiles [Online]. Available: http://www.rackspace.com/cloud/files, Aug. 2015.

[30] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[31] L. Granvilliers, "A symbolic-numerical branch and prune algorithm for solving non-linear polynomial systems," *J. Universal Comput. Sci.*, vol. 4, no. 2, pp. 125–146, 1998.

[32] L. Jiao, J. Li, W. Du, and X. Fu, "Multi-objective data placement for multi-cloud socially aware services," in *Proc. IEEE INFOCOM*, 2014, pp. 28–36.

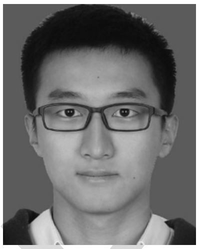
[33] F. Zhang, J. Cao, K. Li, S. U. Khan, and K. Hwang, "Multi-objective scheduling of many tasks in cloud platforms," *Future Gener. Comput. Syst.*, vol. 37, pp. 309–320, 2014.

[34] F. Zhang, J. Cao, W. Tan, S. U. Khan, K. Li, and A. Y. Zomaya, "Evolutionary scheduling of dynamic multitasking workloads for Big-data analytics in elastic cloud," *IEEE Trans. Emerging Topics Comput.*, vol. 2, no. 3, pp. 338–351, Sep. 2014.

[35] O. Khan, R. Burns, J. Plank, W. Pierce, and C. Huang, "Rethinking erasure codes for cloud file systems: Minimizing I/O for recovery and degraded reads," in *Proc. 10th USENIX Conf. File Storage Technol.*, 2012, pp. 251–264.

[36] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, S. Yekhanin, et al., "Erasure coding in windows azure storage," in *Proc. USENIX Conf. Annu. Techn. Conf.*, 2012, vol. 12, p. 2.

**Maomeng Su** received the BE degree from University of Science and Technology Beijing, China, in 2011. He is currently working toward the PhD degree with the Department of Computer Science and Technology, Tsinghua University, Beijing, China. He is currently working on in-memory key-value stores over new datacenter networks. His research interests include cloud storage systems, distributed systems, new datacenter networks, Remote Direct Memory Access (RDMA), and key-value store systems.



**Lei Zhang** is a final year undergraduate student with the Department of Computer Science and Technology at Tsinghua University, Beijing, China, and will receive the BE degree in 2015. He is currently working on storage system and algorithm optimization of cloud computing. His research interests include distributed system, storage system, reliable computing, grid computing, and parallel computing.

**Yongwei Wu** received the PhD degree in applied mathematics from the Chinese Academy of Sciences in 2002. He is currently a professor in computer science and technology at Tsinghua University of China. His research interests include parallel and distributed processing, and cloud storage. He has published more than 80 research publications and has received two Best Paper Awards. He is currently on the editorial board of the *International Journal of Networked and Distributed Computing and Communication of China Computer Federation*. He is a member of the IEEE.

**Kang Chen** received the PhD degree in computer science and technology from Tsinghua University, Beijing, China, in 2004. Currently, he is an associate professor of computer science and technology at Tsinghua University. His research interests include parallel computing, distributed processing, and cloud computing.

**Keqin Li** is a SUNY distinguished professor of computer science. His current research interests include parallel computing and high-performance computing, distributed computing, energy-efficient computing and communication, heterogeneous computing systems, cloud computing, big data computing, CPU-GPU hybrid and cooperative computing, multicore computing, storage and file systems, wireless communication networks, sensor networks, peer-to-peer file sharing systems, mobile computing, service computing, Internet of things and cyber-physical systems. He has published more than 350 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He is currently or has served on the editorial boards of *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Computers*, *IEEE Transactions on Cloud Computing*, *Journal of Parallel and Distributed Computing*. He is a fellow of the IEEE.