# A Reliable and Secure Cloud Storage Schema Using Multiple Service Providers

Haiping Xu and Deepti Bhalerao
Computer and Information Science Department
University of Massachusetts Dartmouth, North Dartmouth, MA 02747, USA
{hxu, dbhalerao}@umassd.edu

*Abstract*—**Despite the many advantages provided by cloud-based storage services, there are still major concerns such as security, reliability and confidentiality of data stored in the cloud. In this paper, we propose a reliable and secure cloud storage schema using multiple service providers. Different from existing approaches to achieving data reliability using redundancy at the server side, we propose a reliable and secure cloud storage schema that can be implemented at the client side. In our approach, we view multiple cloud-based storage services as virtual independent disks for storing redundant data encoded using erasure codes. Since each independent cloud service provider has no access to a user's complete data, the data stored in the cloud would not be easily compromised. Furthermore, the failure or disconnection of a service provider will not result in the loss of a user's data as the missing data pieces can be readily recovered. To demonstrate the feasibility of our approach, we developed a prototype cloud-based storage system that breaks a data file into multiple data pieces, generates an optimal number of checksum pieces, and uploads them into multiple cloud storages. Upon the failure of a cloud storage service, the application can quickly restore the original data file from the available pieces of data. The experimental results show that our approach is not only secure and fault-tolerant, but also very efficient due to concurrent data processing.**

***Keywords-Cloud storage; reliability; data security; erasure codes; cloud service provider; integer linear programming.***

## I. INTRODUCTION

As an ever-growing data storage solution, cloud-based storage services have become a highly practical way for both people and businesses to store their data online. The pay-as-per-use model of cloud computing eliminates the upfront commitment from cloud users; thereby it allows users to start small businesses quickly, and increase resources only when they are needed. However, since data storage locations and security measures at the server site are typically unknown, most of the users have not yet become comfortable with exploiting the full potential of the cloud. Many incidents happened recently have made users question the reliability of cloud storage services. For example, in May 2014, Adobe's ID service went down, leaving Creative Cloud users locked out of their software and account for over 24 hours [1]. In early 2013, Dropbox service had a major cloud outage that kept users offline and unable to synchronize using their desktop apps for more than 15 hours [2]. Prolonged cloud data service outages and security concerns can be fatal for businesses with data critical domains such as healthcare, banking and finance. Today, almost all the cloud service providers (CSP) have implemented fault-tolerant mechanisms at their server sides to recover original data from service failure or data corruption. Such mechanisms are suitable at the time of scheduled maintenance or for a small number of hard disk failures. However, they are of no use for the end users to ensure the reliability and security of their cloud data when major cloud services fail or the cloud services have been compromised. Hence, to achieve high reliability and security of critical data, users should not depend upon a single cloud service provider. In this paper, we propose an approach that can provide security and fault tolerance to the user's data from the client side. In our approach, we decompose an original data file into multiple data pieces, and generate checksum pieces using erasure codes [3]. The pieces of data are spread across multiple cloud services, which can be retrieved and combined to recover the original file. We achieve data redundancy in our approach using erasure codes at the software level across multiple cloud service providers. Therefore, the original data can be recovered even when there is a cloud outage where some cloud service fails completely. Using this approach, user's data would not be easily compromised by unauthorized access and security breach, as no single cloud service has the complete knowledge of user's data. Thus, users could have the sole control of their cloud data, and do not need to rely on the security measures provided by cloud service providers. Finally, to improve the network performance of our approach, we adopt the multithreading technology, and fully utilize the network bandwidth in order to minimize the time required to access data over the cloud.

There have been many research efforts on using erasure codes at the server side to make cloud storage service reliable. Huang *et al.* proposed to use erasure codes in Windows Azure storage [4]. They introduced a new set of codes for erasure codes called Local Reconstruction Codes (LRC) that could reduce the number of erasure coding fragments required for data reconstruction. Gomez *et al.* introduced a novel persistency technique that leverages erasure codes to save data in a reliable fashion in Infrastructure as a Service (IaaS) clouds [5]. They presented a scalable erasure coding algorithm that could support a high degree of reliability for local storage with the cost of low computational overhead and a minimal amount of communication. Khan *et al.* provided guidance for deploying erasure coding in cloud file systems to support load balance and incremental scalability in data centers [6]. Their proposed approach can prevent correlated failures with data

(DOI Reference Number: 10.18293/SEKE2015-045)

loss and mitigate the effect of any single failure on a data set or an application. Although the above approaches can significantly enhance the reliability of cloud data at data centers, they provide no support for end users to deal with failures or cloud outage of the service providers. Different from the existing approaches, we apply erasure-coding techniques at the application level using multiple cloud service providers. By deploying user's encoded redundant data across multiple cloud storage services, our approach is fault tolerant for cloud storage when any of the cloud services fails.

There is also a considerable amount of work on securing cloud data, to which this work is closely related. Santos *et al.* proposed a secure and trusted cloud computing platform (TCCP) for IaaS providers such as Amazon EC2 [7]. The platform provides a closed box execution environment that guarantees confidential execution of guest virtual machines on a cloud infrastructure. Hwang and Li proposed to use data coloring and software watermarking techniques to protect shared cloud data objects [8]. Their approach can effectively prevent data objects from being damaged, stolen, altered, or deleted, and users may have their sole access to their desired cloud data. The existing approaches to securing cloud data typically assume that the cloud service providers are trustable and they can prevent physical attacks to their servers. However, this might not be true in reality, as service providers typically tend to collect users' cloud data for their commercial purposes such as targeted adverting. Furthermore, there have been many incidents that cloud service providers were compromised by either internal or external hackers, and thousands of users' critical data were compromised. Therefore, merely relying on service providers' security mechanisms is not a feasible solution for both people and businesses to store their critical data in the cloud. It is required that users should be allowed to apply security mechanisms to their own data at the client side. Different from the aforementioned methods to securing cloud data at the server side, our approach does not rely on any security measures supported by the service providers. Instead, the cloud storage application running at the client side can split users' data into pieces, encode them using erasure codes, and distribute them to multiple service providers. As no single CSP has its access to a user's entire data, user's data are much securer than those stored with a single cloud service.

In this paper, we extend the methodology and results of a preliminary study on secure and fault-tolerant model of cloud information storage [9]. In the previous work, we followed the RAID (Redundant Array of Independent Disks) approach to encode user's data using XOR parity, and developed a hierarchical colored Petri nets (HCPN) model for secure and fault-tolerant cloud information storage systems. In this paper, we adopted erasure codes to achieve fault tolerance for cloud data, and presented a detailed design for a reliable and secure cloud storage schema. To demonstrate the effectiveness of our proposed approach, we implemented a prototype using three major cloud service providers (i.e., Amazon, Google and Dropbox), which allows users to securely, reliably and efficiently store their critical data in the cloud.

## II. Reliable and Secure Cloud Data Storage

To address the aforementioned major issues in cloud storage services, we propose a reliable and secure cloud storage schema using multiple CSPs. Figure 1 shows a framework for such a system. The major component of the system is the cloud storage application that uses erasure codes to encode and decode file pieces at the client side, and upload and download encoded file pieces concurrently at multiple cloud services. As shown in the figure, when a user wants to upload a file into the cloud, the application first splits the file into multiple data pieces, say $n$ pieces, and then encode them into an optimal number of $m$ checksum pieces using the erasure coding technique. Once the data pieces and checksum pieces are ready, they are concurrently uploaded into multiple cloud storages maintained by different CSPs, noted as *CSP_1*, *CSP_2*, …, and *CSP_N* in Fig. 1. As none of the CSPs has the complete knowledge about the user data, this approach can effectively defend against data breach from any single CSP.

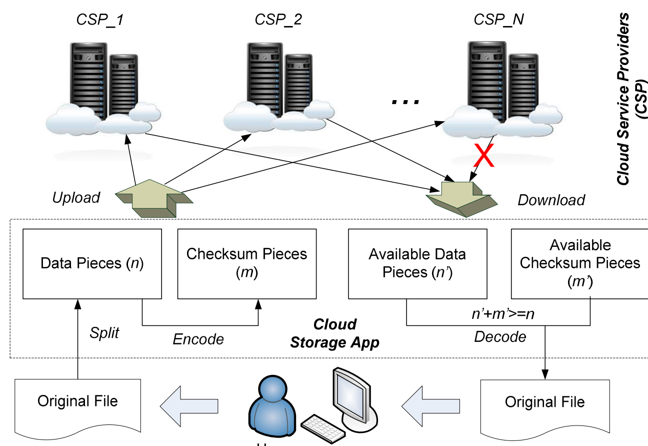


Figure 1. A framework for reliable and secure cloud storage systems

On the other hand, when a user wants to download a stored file, the application will first try to download the $n$ data pieces from the multiple cloud storages concurrently. If all data pieces are available, they can be efficiently combined into the original file without any additional decoding process. However, in the case when one or more service provider fails, the application must automatically download all available data pieces ($n'$) and available checksum pieces ($m'$). As long as $n' + m' \geq n$, due to the erasure coding technique, the application can always successfully decode the missing data pieces using the available pieces of data, and restore the original file. Note that the checksum pieces serve as the redundancy of the original file, which makes our approach reliable and fault tolerant.

## III. Erasure Codes and Reed-Solomon Coding

### A. *Erasure Codes*

In early days, fault tolerance of cloud data is commonly achieved through simple data replication. Multiple copies of original data have to be maintained on different cloud servers in order to make data more reliable. However, data replication now becomes highly unfeasible due to its low space efficiency and the ever-increasing amount of cloud data. Erasure codes, also known as forward error correction (FEC) codes, manage to overcome the disadvantages of the data replication approach, and can achieve a high degree of fault tolerance with a much lower cost of physical storage [3]. An erasure code takes $n$ data

words and transforms them into $m$ code words such that any $n$ out of ($n$+$m$) words are enough to recover the original $n$ data words. Erasure codes use a mathematical function to convert original data words into encoded words and to recover them back. They can be very efficient in providing fault tolerance for large quantities of data, hence they are quite suitable for large-scale cloud storage systems.

Data redundancy through parity codes represents the simplest form of erasure codes, which overcomes the drawback of data replication. RAID-5 is the most commonly used technique that uses parity codes. It calculates parities from the original data to achieve fault tolerance. However, this technique is typically used by CSPs at the hardware level, and very few research efforts attempted to apply the RAID concept at the software level to resolve issues related to the major data failures of a service provider, which actually have become quite common nowadays [9].

### B. Reed-Solomon Coding for Cloud Based Storage

Use of error-correction codes for redundancy has become prevalent due to its various advantages. Reed-Solomon (RS) coding is a type of optimal erasure codes, which follows the basic error-correction techniques. There are many different ways to implement error-correction using erasure codes, but RS technique is a good compromise between efficiency and complexity [10]. Traditionally, RS technique has been used in various applications such as error-correction in CD-ROM and DVDs, satellite communications, digital television, and wireless or mobile communications [11]. The use of RS technique to provide fault tolerance over the cloud is a fairly new idea. Our approach to distributing data and checksum pieces with multiple cloud services could build a RAID-like system with less storage overhead and more flexibility in the degree of fault tolerance for the stored data. Here we first briefly introduce the RS coding approach. Let there be $n$ data pieces, we encode all data pieces using RS algorithm into $m$ checksum pieces such that out of ($n$+$m$) pieces, any $n$ pieces are enough to recover the original $n$ data pieces. If the ($n$+$m$) pieces of data are distributed over ($n$+$m$) devices, this algorithm can be used to handle $m$ failures of the devices.

To simplify matters, we assume each data piece is an unsigned byte ranged from 0 to 255. In order to calculate the checksum bytes, we first create an $(m+n)\times n$ Vandermonde matrix $A$, where the $i$, $j$-th element of $A$ is defined to be $i^j$ [11]. By this definition, when $m$ rows are deleted from $A$, the newly formed matrix is invertible. Then we derive the information dispersal matrix $B$ from $A$ using a sequence of elementary matrix transformation. The information dispersal matrix $B$ is defined as in Eq. (1), where $I$ is an $n\times n$ identity matrix, and $F$ is an $m\times n$ matrix. Note that since elementary matrix transformation does not change the rank of a matrix and each row in $A$ is linearly independent, the information dispersal matrix $B$ maintains the property that when $m$ rows are deleted from $B$, the newly formed matrix is invertible.

$$B=\begin{bmatrix} I \\ F \end{bmatrix} \quad (1) \qquad BD=\begin{bmatrix} I \\ F \end{bmatrix} D = E, \text{ where } E=\begin{bmatrix} D \\ C \end{bmatrix} \quad (2)$$

Let $D$ be a vector of $n$-byte data $d_0$, $d_1$, ..., $d_{n-1}$, and $C$ be a vector of $m$-byte checksum $c_0$, $c_1$, ..., $c_{m-1}$. With the information dispersal matrix $B$, we can calculate the checksum vector $C$ from the data vector $D$ as in Eqs. (3), where $f_{i,j}$ ($0 \le i \le m-1$, $0 \le j \le n-1$) are elements of the $m\times n$ matrix $F$. Based on the calculation of $C$, Eq. (2) must hold.

$$\begin{aligned} c_0 &= f_{0,0}*d_0 + f_{0,1}*d_1 + \ldots + f_{0,n-1}*d_{n-1} \\ c_1 &= f_{1,0}*d_0 + f_{1,1}*d_1 + \ldots + f_{1,n-1}*d_{n-1} \\ &\ldots \\ c_{m-1} &= f_{m-1,0}*d_0 + f_{m-1,1}*d_1 + \ldots + f_{m-1,n-1}*d_{n-1} \end{aligned} \quad (3)$$

Now suppose $k$ bytes, where $k \le m$, are missing from vector $D$. By deleting the missing $k$ elements from $D$ as well as any $m$-$k$ elements from $C$, we derive a new $n$-byte vector $E'$ as in Eq. (4), where $D'$ is a ($n$-$k$)-byte vector $d_0', d_1', \ldots, d_{n-k-1}'$, and $C'$ is a $k$-byte vector $c_0', c_1', \ldots, c_{k-1}'$. Similarly, in Eq. (2), by deleting $m$ rows from $B$ that correspond to the deleted rows in $E$, we drive an $n\times n$ matrix $B'$ as defined in Eq. (5), where $I'$ is an $(n-k)\times n$ matrix, and $F'$ is a $k\times n$ matrix. The matrix $B'$ must be invertible as we have mentioned, and Eq. (6) must hold.

$$E'=\begin{bmatrix} D' \\ C' \end{bmatrix} \quad (4) \qquad B'=\begin{bmatrix} I' \\ F' \end{bmatrix} \quad (5) \qquad B'D=\begin{bmatrix} I' \\ F' \end{bmatrix} D = \begin{bmatrix} D' \\ C' \end{bmatrix} \quad (6)$$

By calculating the inverse matrix $G = B'^{-1}$ using Gaussian elimination method, we can recover the data vector $D$ as in Eqs. (7), where $g_{i,j}$ ($0 \le i \le n-1$, $0 \le j \le n-1$) are elements of the $n\times n$ matrix $G$.

$$\begin{aligned} d_0 &= g_{0,0}*d_0' + g_{0,1}*d_2' +,\ldots,+ g_{0,n-k-1}*d_{n-k-1}' + \\ &\quad g_{0,n-k}*c_0' + g_{0,n-k+1}*c_1' +,\ldots,+ g_{0,n-1}*c_{k-1}' \\ d_1 &= g_{1,0}*d_0' + g_{1,1}*d_2' +,\ldots,+ g_{1,n-k-1}*d_{n-k-1}' + \\ &\quad g_{1,n-k}*c_0' + g_{1,n-k+1}*c_1' +,\ldots,+ g_{1,n-1}*c_{k-1}' \\ &\ldots \\ d_n &= g_{n-1,0}*d_0' + g_{n-1,1}*d_2' +,\ldots,+ g_{n-1,n-k-1}*d_{n-k-1}' + \\ &\quad g_{n-1,n-k}*c_0' + g_{n-1,n-k+1}*c_1' +,\ldots,+ g_{n-1,n-1}*c_{k-1}' \end{aligned} \quad (7)$$

Once the $n$-byte vector $D$ is restored, the $m$-byte vector $C$ can be recalculated as in Eqs. (3). Note that implementation of the RS algorithm for data files requires to perform computations on binary words of a fixed length $w$. For example, when the binary word is a byte, $w$ equals 8. To ensure that the RS algorithm works correctly for fixed-size words, all arithmetic operations must be performed over Galois Fields with $2^w$ elements denoted as $GF(2^w)$ [11]. A Galois field $GF(2^w)$ is also known as a finite field which contains finitely many elements, namely 0, 1, ..., $2^w$-1. Arithmetic operations performed over Galois Fields will result in finite values in $GF(2^w)$. As such, all arithmetic operations mentioned in this section, including the matrix inverse, encoding and recovery of data, must be calculated using Galois Fields arithmetic.

## IV. Optimal Number of Checksum Pieces

### A. Calculating the Optimal Number of Checksum Pieces

In order to achieve the highest space efficiency in our approach, we propose a procedure to compute the minimal number of checksum pieces that allows the failures of multiple cloud service providers. Let $N$ be the number of service

providers, $\Gamma = \{1, 2, ..., N\}$, and $M$ be the maximal number of services allowed to fail or become unavailable at the same time, where $1 \le M \le N$-1. We define a failure set $\Phi$ as follows:

$\Phi \in \mathcal{P}(\Gamma)$, where $\mathcal{P}(\Gamma)$ is the power set of $\Gamma$, and $|\Phi| \le M$.

The set of available CSPs $\Omega$ corresponding to $\Phi$ can be defined as in Eq. (8).

$$\Omega = \Gamma - \Phi \tag{8}$$

Let the number of data pieces of a file be $n$. In order to distribute $n$ data pieces evenly over $N$ cloud service providers, we calculate the number of data pieces $n_1$, $n_2$, ..., and $n_N$ stored at $CSP1$, $CSP2$, ..., and $CSP_N$, respectively, as in Eq. (9).

$$n_i = \begin{cases} \lceil n/N \rceil & when \quad i = 1 \\ \left\lceil (n - \sum_{j=1}^{i-1} n_j)/(N - i + 1) \right\rceil & when \quad 1 < i < N \\ n - \sum_{j=1}^{N-1} n_j & when \quad i = N \end{cases} \tag{9}$$

where $n = n_1 + n_2 + ... + n_N$. Eq. (9) allows even distribution of $n$ data pieces over $N$ cloud service providers such that $|n_i - n_j| \le 1$ for $1 \le i, j \le N$. For example, when $N = 3$ and $n = 7$, the number of data pieces distributed over three cloud service providers $CSP1$, $CSP2$, and $CSP3$ will be 3, 2, 2, respectively.

As a major requirement for fault tolerance, when up to $M$ CSPs become unavailable, the original data must be recovered from the remaining CSPs from the available set $\Omega$. Let $m$ be the number of checksum pieces required, and $m_1$, $m_2$, ..., $m_N$ are the numbers of checksum pieces distributed over $CSP1$, $CSP2$, ..., and $CSP_N$, respectively. Obviously, we have $m = m_1 + m_2 + ... + m_N$. To calculate the minimal number of checksum pieces $m$, we can solve the integer linear programming problem as defined in Eq. (10).

$$\begin{aligned} &\texttt{minimize} \quad \sum_{i=1}^{N} m_i \\ &\texttt{subject to} \quad \text{for each failure set } \Phi \\ &\qquad \sum_{i \in \Omega} m_i \ge \sum_{j \in \Phi} n_j \\ &\qquad \text{where } \Phi \in \mathcal{P}(\Gamma) \text{ and } |\Phi| = M \end{aligned} \tag{10}$$

Note that a solution to the above optimal problem automatically satisfies the cases when $|\Phi| < M$. The space efficiency $e$ of a solution can be calculated as in Eq. (11).

$$e = 1 - m/(n + m), \text{ where } n = \sum_{i=1}^{N} n_i \text{ and } m = \sum_{i=1}^{N} m_i \tag{11}$$

As an example, let $N = 3$ and $M = 1$, the integer linear programming problem can be simplified as in Eq. (12).

$$\begin{aligned} &\texttt{minimize} \quad m_1 + m_2 + m_3 \\ &\texttt{subject to} \quad m_1 + m_2 >= n_3 \quad // \text{ when } \Phi = \{3\} \\ &\qquad m_2 + m_3 >= n_1 \quad // \text{ when } \Phi = \{1\} \\ &\qquad m_1 + m_3 >= n_2 \quad // \text{ when } \Phi = \{2\} \end{aligned} \tag{12}$$

Table 1 shows the optimal solutions and their space efficiency for the above example with $n$ ranges from 2 to 14. For instance, when $n = 8$ ($n_1 = 3$, $n_2 = 3$, $n_3 = 2$), the optimal solution is $m_1 = 1$, $m_2 = 1$, and $m_3 = 2$, and the space efficiency $e$ = 1 - 4/(8+4) = 0.6667. In this case, if any service provider becomes unavailable, the missing 4 pieces of data can always be recovered from the remaining data and checksum pieces stored at the other two CSPs.

Table 1. Optimal number of checksum pieces and space efficiency

| Data Pieces (n) | ($n_1$, $n_2$, $n_3$) | ($m_1$, $m_2$, $m_3$) | Checksum Pieces (m) | Space Efficiency (e) |
|---|---|---|---|---|
| 2 | (1, 1, 0) | (0, 0, 1) | 1 | 0.6667 |
| 3 | (1, 1, 1) | (0, 0, 1) | 2 | 0.6000 |
| 4 | (2, 1, 1) | (0, 1, 1) | 2 | 0.6667 |
| 5 | (2, 2, 1) | (1, 1, 1) | 3 | 0.6250 |
| 6 | (2, 2, 2) | (1, 1, 1) | 3 | 0.6667 |
| 7 | (3, 2, 2) | (1, 2, 1) | 4 | 0.6364 |
| 8 | (3, 3, 2) | (1, 1, 2) | 4 | 0.6667 |
| 9 | (3, 3, 3) | (2, 2, 1) | 5 | 0.6429 |
| 10 | (4, 3, 3) | (1, 2, 2) | 5 | 0.6667 |
| 11 | (4, 4, 3) | (2, 2, 2) | 6 | 0.6471 |
| 12 | (4, 4, 4) | (2, 2, 2) | 6 | 0.6667 |
| 13 | (5, 4, 4) | (2, 3, 2) | 7 | 0.6500 |
| 14 | (5, 5, 4) | (3, 3, 2) | 7 | 0.6667 |

### *B. Distibution of Data and Checksum Pieces over CSPs*

When dealing with a file with $k$ bytes, if $k$ is not a multiple of $n$, we first need to append $r$ bytes with random values to the end of the file such that $((k + r) \bmod n) = 0$. Then we group the $(k+r)$ bytes into $n$ data pieces so that each of them contains $(k+r)/n$ bytes. By applying Eq. (9) and Eq. (10), we calculate the distribution of the $n$ data pieces and the optimal number of checksum pieces. Finally, using the equations described in Section III.B, we can calculate the checksum pieces. Figure 2 shows an example of file distribution at service providers $CSP1$, $CSP2$ and $CSP3$ when $N = 3$, $M$ =1, $n = 8$ and $m = 4$.

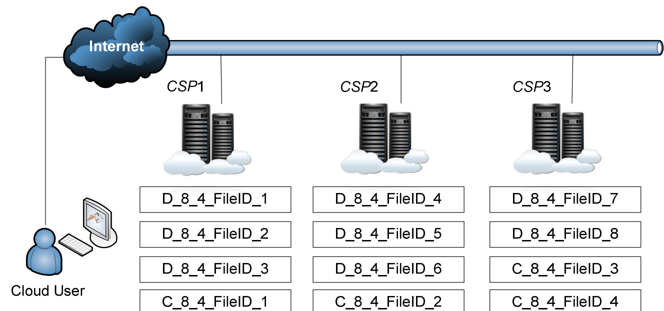


Figure 2. Distribution of data and checksum pieces at three CSPs

As shown in Fig. 2, we distribute 3, 3, 2 data pieces (denoted by the file names starting with the letter "D") over $CSP1$, $CSP2$ and $CSP3$, respectively. Based on the optimal solution given in Table 1, we also distribute 1, 1, 2 checksum pieces (denoted by the file names starting with the letter "C") over $CSP1$, $CSP2$ and $CSP3$, respectively. When any of the service providers fails, the original data can be recovered from the remaining 8 pieces of data using Eq. (7). It is worth noting that by the definition of the RS coding technique, when up to 4 pieces of data from multiple CSPs are missing or corrupted, the original file can still be recovered using Eq. (7).

## V. Case Study

To demonstrate the feasibility of our proposed approach, we developed a prototype secure and reliable cloud storage application in Java. We adopt three different cloud services supported by major CSPs to store our data pieces and checksum pieces in the cloud. The selected cloud services are Amazon S3, Google App Engine, and Core Dropbox APIs

with free user accounts. All experiments have been conducted with excellent Internet connections at University of Massachusetts Dartmouth, where the download speed was around 160 Mbps (~20MB/s) and the upload speed was around 400 Mbps (~50MB/s). Therefore, the network connection at the client side will not become a bottleneck for all of our experiments. As shown in Fig. 3, the user interface of the application allows one to select a file to upload into the cloud. After choosing the number of data pieces ($n$), the optimal number of checksum pieces ($m$) can be automatically calculated using integer linear programming. By clicking on the "Encode and Upload" button, the selected file is divided into $n$ data pieces, and the application automatically encodes them into $m$ checksum pieces. Once all pieces of data become ready, they are uploaded into the three selected cloud storage services using multithreading techniques. The message box in the user interface displays the encoding time, the uploading time and the total processing time.

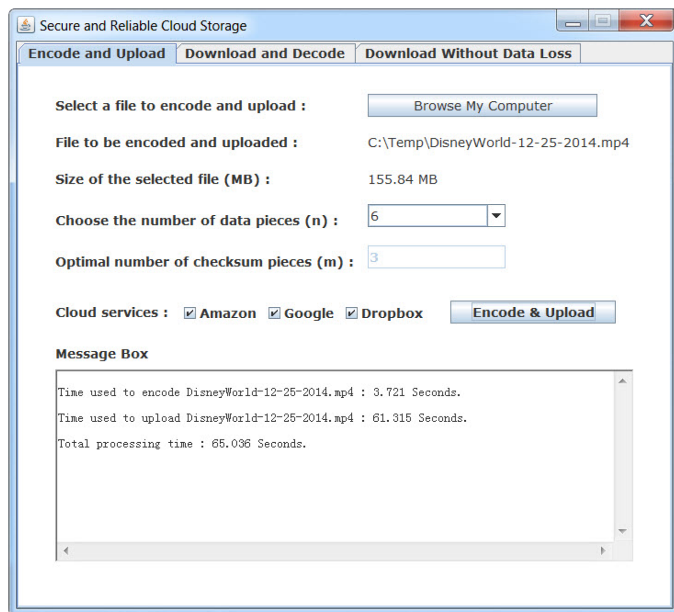


Figure 3. Encode and upload a file to multiple cloud storage services

Figure 4 shows the user interface for downloading and decoding an uploaded file in the cloud. As shown in the figure, a user first selects a file from the list of uploaded files, then chooses at least two cloud service providers as the maximal number of failed cloud services $M$ equals 1. By clicking on the "Download and Decode" button, the available file pieces are concurrently downloaded to the local computer, where the original file is recovered using RS coding techniques. Similarly, as shown in Fig. 4, the message box in the user interface displays the downloading time, the decoding time and the total processing time, as well as the location of the downloaded file on the user's local computer.

To analyze the performance of our approach, we selected a video file with a file size of 156 MB. Figure 5 shows the encoding and uploading time vs. the number of data pieces set by the user. From the figure, we can see that when we increase the number of data pieces from 2 to 8, the uploading time drops down significantly; while the encoding time has slightly increased. The significant performance improvement for uploading is due to the use of multithreading techniques; however, the increased number of data pieces along with more checksum pieces result in more overhead for encoding. When the number of data pieces $n$ is further increased, the uploading time dramatically goes up. Based on our further experiments with the cloud service providers, the concurrent processing capabilities of the service providers as well as their bandwidths become a major issue when the number of concurrent uploading reaches 5. Note that when $n$ = 10, the optimal number of checksum pieces $m$ = 5, so the number of concurrent uploading to each CSP is 5.

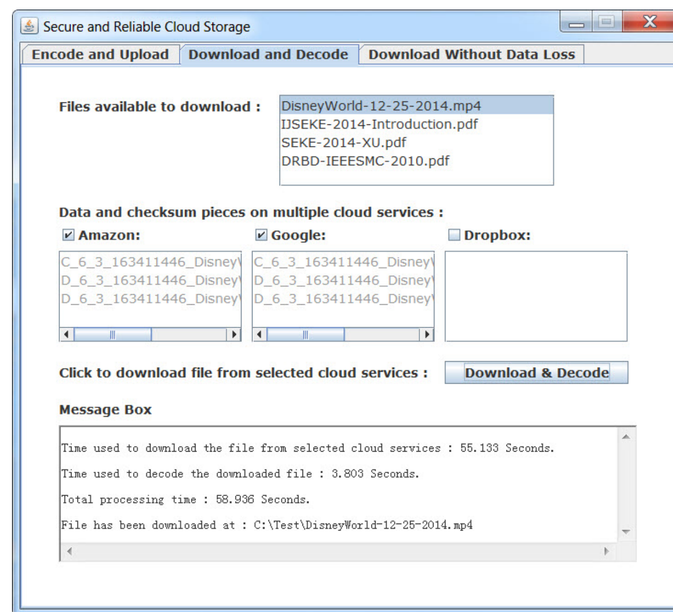


Figure 4. Download and decode a file from clouds with a failed service

Figure 6 shows the downloading and decoding time vs. the number of data pieces set by the user. From the figure, we can see that when we increase the number of data pieces from 2 to 9, the downloading time drops down significantly; while the decoding time has slightly increased. Similar to the case of uploading, the significant performance improvement for downloading is also due to the multithreading techniques, and the increased number of data pieces along with more checksum pieces result in more overhead for decoding. When the number of data pieces $n$ is further increased, the downloading time goes up slightly, which it is not as bad as in the uploading case with dramatic performance change. This is because major cloud service providers typically put more restrictions on their upload bandwidths than their download bandwidths, especially for free user accounts.

From the above experimental results, we can see that both the uploading and downloading time can be significantly reduced by selecting a reasonable number of data pieces. For example, when a file size is between 100 to 200 MB, based on our experiments, the number of data pieces should normally be set to 8 as long as the network bandwidth is sufficient. According to Table 1, when $n$ = 8, the optimal number of checksum pieces $m$ = 4. In this case, the space efficiency $e$ reaches its highest value 0.6667. It is worth noting that when no service provider fails, the application only needs to download the data pieces, and no checksum pieces are needed

for restoring the original file. In this case, the downloading time can be further reduced, and the decoding time becomes merely the time needed to combine the data pieces into the original file. Therefore, in a normal case with no failure of service providers, the overall performance for file retrieval will be better than the results demonstrated in Fig. 6.

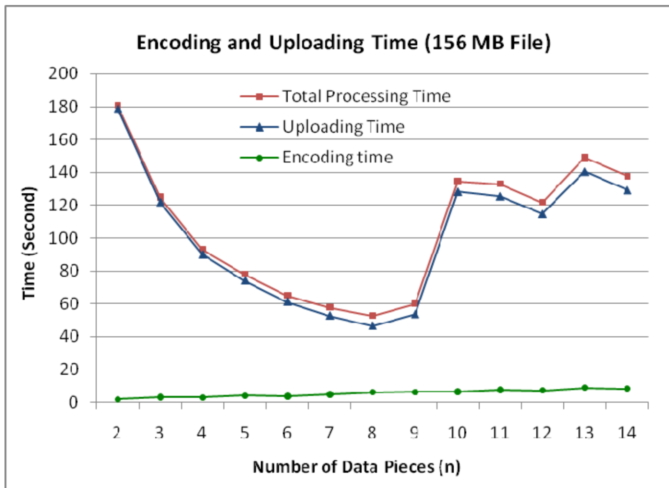


Figure 5. Encoding & uploading time vs. number of data pieces

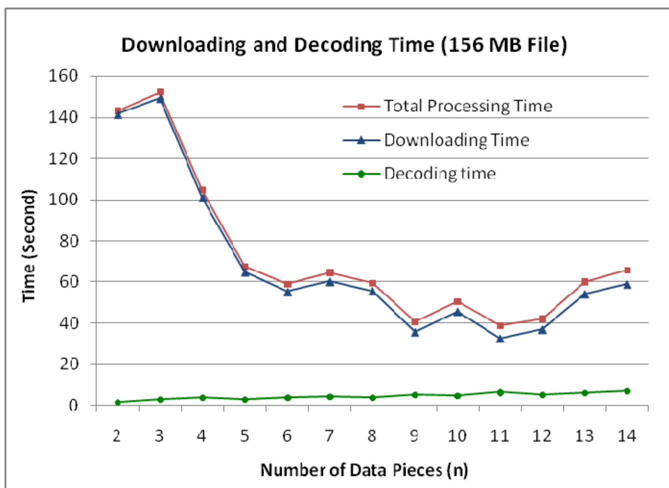


Figure 6. Downloading and decoding time vs. number of data pieces

## VI. CONCLUSIONS AND FUTURE WORK

In this paper, we addressed three major issues with cloud storage, namely reliability, security and performance. Instead of achieving data reliability using redundancy at the server side, we presented a reliable and secure cloud storage schema for end users. In our approach, we view multiple cloud storage services as virtual disks, and upload redundant data files into multiple cloud storages. The redundant data files are calculated using erasure codes techniques, which allow multiple failures of the data pieces. By forming an optimal problem for calculating the number of checksum pieces, we can achieve the best space efficiency in our approach. Furthermore, we divide the user data into pieces, and distribute them across multiple cloud services; therefore, no single CSP can understand the uploaded user data. As a result, our approach can effectively protect user data from unauthorized access in the cloud, and provide security at the software level for the end users. Finally, the experimental results show that due to concurrent data processing, our approach provides very good performance in file uploading and downloading, with the cost of minor overhead for encoding and decoding data.

For future work, we will investigate possible ways to automatically select a suitable number of data pieces based on the network condition and the file size. We will consider other major aspects of cloud data, such as data integrity and confidentiality. For example, it would be feasible to adopt the digital signature technique to verify the integrity of the data stored in the cloud to ensure they were not altered by the service providers. Furthermore, when large cloud files are involved, the overhead for encoding and decoding may become a concern. To improve the overall performance in this case, we need to look into more advanced techniques for erasure codes, such as regenerating codes and non-MDS codes [3]. Finally, we will attempt to integrate our approach with cloud-based big data analysis for reliable and secure data stored in the cloud. This may also be considered as a worthy future direction.

## REFERENCES


[1] S. Yegulalp, "Adobe Creative Cloud Crash Shows that No Cloud is Too Big to Fail," *InfoWorld*, May 16, 2014. Retrieved on March 7, 2015 from http://www.infoworld.com/article/2608200/cloud-computing/adobe-creative-cloud-crash-shows-that-no-cloud-is-too-big-to-fail.html

[2] C. Talbot, "Dropbox Outage Represents First Major Cloud Outage of 2013," *Talkin'Cloud*, Jan 15, 2013. Retrieved on May 18, 2014 from http://talkincloud.com/cloud-storage/dropbox-outage-represents-first-major-cloud-outage-2013

[3] J. S. Plank, "Erasure Codes for Storage Systems: A Brief Primer," *Login: The USENIX Magzine*, www.usenix.org, December 2013, Vol. 38, No. 6, pp. 44-50.

[4] C. Huang, H. Simitci, Y. Xu *et al.*, "Erasure Coding in Windows Azure Storage," *Proceedings of the 2012 USENIX Annual Technical Conference*, Boston, MA, USA, pp. 15-26 , June 13-15, 2012.

[5] L. B. Gomez, B. Nicolae, N. Maruyama, F. Cappello and S. Matsuoka, "Scalable Reed-Solomon-based Reliable Local Storage for HPC Applications on IaaS Clouds," *Proceedings of the 18th International Euro-Par Conference on Parallel Processing (Euro-Par'12)*, Rhodes, Greece, pp. 313-324, August 2012.

[6] 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," *Proceedings of the 10th USENIX Conference on File and Storage Technologies (FAST-2012)*, San Jose, CA, USA, pp. 20-33, February 2012.

[7] N. Santos, K. Gummadi, and R. Rodrigues, "Towards Trusted Cloud Computing," *Proceedings of the Workshop on Hot Topics in Cloud Computing (HotCloud09)*, Article No. 3, San Diego, CA, June 15, 2009.

[8] K. Hwang and D. Li, "Trusted Cloud Computing with Secure Resources and Data Coloring," *IEEE Internet Computing*, Vol. 14, No. 5, pp. 14-22, 2010.

[9] D. Fitch and H. Xu, "A RAID-Based Secure and Fault-Tolerant Model for Cloud Information Storage," *International Journal of Software Engineering and Knowledge Engineering (IJSEKE)*, Vol. 23, No. 5, 2013, pp. 627-654.

[10] C. K. Clarke, "Reed-Solomon Error Correction," *R&D White Paper*, British Broadcasting Corporation, July 2002.

[11] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*, North-Holland Mathematical Library, Amsterdam, London, New York, Tokyo, 1977.