Article

Data Chunks Placement Optimization for Hybrid Storage Systems

Agil Yolchuyev * and Janos Levendovszky

Department of Networked Systems and Services, Budapest University of Technology and Economics, Magyar Tudosok krt. 2, 1117 Budapest, Hungary; levendov@hit.bme.hu
* Correspondence: yolchuyev@hit.bme.hu

Abstract: "Hybrid Cloud Storage" (HCS) is a widely adopted framework that combines the functionality of public and private cloud storage models to provide storage services. This kind of storage is especially ideal for organizations that seek to reduce the cost of their storage infrastructure with the use of "Public Cloud Storage" as a backend to on-premises primary storage. Despite the higher performance, the hybrid cloud has latency issues, related to the distance and bandwidth of the public storage, which may cause a significant drop in the performance of the storage systems during data transfer. This issue can become a major problem when one or more private storage nodes fail. In this paper, we propose a new framework for optimizing the data uploading process that is currently used with hybrid cloud storage systems. The optimization is concerned with spreading the data over the multiple storages in the HCS system according to some predefined objective functions. Furthermore, we also used Network Coding technics for minimizing data transfer latency between the receiver (private storages) and transmitter nodes.

Keywords: cloud computing; cloud storages; optimization algorithms; network coding

Citation: Yolchuyev, A.; Levendovszky, J. Data Chunks Placement Optimization for Hybrid Storage Systems. *Future Internet* **2021**, *13*, 181. https://doi.org/10.3390/fi13070181

Academic Editor: Mamoun Alazab

Received: 10 June 2021
Accepted: 4 July 2021
Published: 11 July 2021

1. Introduction

Cloud computing has become a trend in technology, due to providing massive, large scale services for solving complex information technology (IT) problems. These services present a wide range of advantages, ranging from organizational data management over the different storages systems to providing secure access to different recourses, as well as administrate them by using multiple groups.

As part of the cloud technologies, cloud storage systems were always an attractive alternative to private distributed storage systems, due to their higher durability and their lower storage cost. Nonetheless, storing the entire data in the cloud brings forth data privacy or vendor-lock issues, which can cause unavailability or, ultimately, the loss of data. For example, an interruption [1] in February 2017 of Amazon S3 in North Virginia showed that even big cloud storage vendors struggle with service failures. To avoid this problem, many organizations investigated the opportunities of integrating the cloud and private distributed storage systems as part of the single uploading scheme. The main problem related to this scheme is having redundancy in the system. The simplest form of redundancy is replication of data chunks over the network. Specifically, for these cases, erasure coding offers better storage efficiency. However, in hybrid cloud storage systems, redundancy must be continually refreshed as nodes fail or leave the system. This involves large data transfers across the network and requires higher recovery bandwidth for remaining robust enough in the case of node failures. Moreover, when the recovery bandwidth increases, the cost of data recovery expands simultaneously. All of this raises a question about the effectiveness of current data recovery models for hybrid cloud storage systems and triggers the idea of having an efficient model for handling the data uploading process.

In this paper, we address the issue of cost-effective redundancy in hybrid storage systems. We build a new framework using "Network Coding" technology on top of the peer-to-peer connection, between HCS nodes and categorized storage systems, in two subgroups:

- Transfer Nodes—Data storages, which keep a combination of the chunks from different objects; and
- Receivers Nodes—Private storages, to keep the object for user access.

If a single receiver node fails and/or is replaced with new storage, we transfer all necessary fragments from transfer nodes to rebuild the object in the storage. Meanwhile, when maintaining redundancy in the system, we seek storages, which we can use for minimizing the amount of the bandwidth for object recovery in the new receiver storage. For this, we introduced a new heuristic for finding optimal data chunks placement.

The paper treats this material in the following structure:

- in Section 2, we provide a brief literature survey,
- in Section 3, we formulate the problem of optimal file upload,
- in Section 4, we present the optimal solution to the problem, and
- in Section 5, we describe the stimulation environments and results.

2. Summary of Results

In this section, we summarize the current state-of-art solutions, in the field of distributed storage/cloud storage systems, together with their pricing models for data storage.

2.1. Background Research for Distributed and Cloud Storage Systems

Throughout the past few decades, erasure coding schemes, like Reed-Solomon codes, were an attractive alternative to replication in distributed storage systems, due to optimal balancing between storage cost and data durability. It was used as the main error-coding method in various papers (e.g., References [2,3]) for reducing repair bandwidth costs. This method, however, has a higher rate of the coding, which is not suitable for many applications and storage systems, like "Network Attached Storage" (NAS). Moreover, erasure coding consumes a significant amount of recourses during the replication of the data over multiple storages [4].

The idea of implementing Network Coding to the distributed storage systems was first proposed by Dimakis, Prabhakaran, and Ramchandran [5]. It has a number of advantages over standard erasure coding methods, like recoding already encoded data without destroying the properties of the code.

The concept of performing erasure and Network Coding to the cloud storage systems was the main research objective for multiple papers, like [6] Reference [7]. Researchers were mostly concerned with the optimal way of storing the data, i.e., with distributing over the multiple cloud vendors [8]. When the data is stored on cloud storages, several "Quality of Service Factors" factors, like availability, durability, and vendor lock-in, are required to be taken into consideration. While striping data with traditional erasure codes performs well for short-term data storages uploading, for long-term storage, it is not always affordable, especially for the cases when there is a redundant storage/storage in the system. A repair operation of the redundant cloud storage node requires retrieving the data over the network from existing surviving cloud nodes. Because cloud vendors have high outgoing (repair) traffic prices, moving an enormous amount of data across clouds is not affordable (see Section 2.2). This triggered researchers to look into cost-effective alternatives of the erasure coding methods for maintaining data redundancy and fault tolerance over cloud storages. Further research on this topic mostly focused on minimizing repair traffic by using Network Coding and applications of it, e.g., "NCCloud" [9], which was built on top of a network-coding-based storage scheme called minimum-storage regenerating (MSR) codes.

2.2. Public Cloud Storage Cost Strategies

Most of the vendors offer a wide array of different cloud storage services, and the price for each of those services varies from storage provider to storage provider. Because of the large variety of cloud storage services, different pricing models are employed by cloud vendors [10]. Nonetheless, most of the pricing models have a common substructure, namely: they charge for the used storage, the used outgoing traffic, and the amount of reading and write operations. Specifically, cloud vendors charge mostly for two types of data activity:

- Data transfer cost [11]: Type of rate which is charging for every GB downloaded from the cloud or moved to another cloud facility.
- Requests (API calls) cost [12]: Type of pricing for data operations within the storage, e.g., copying data or adding new objects. As a rule, all requests are paid, except commands for data deletion.

Furthermore, most cloud providers use a block rate pricing model [13], where data storing prices decrease with storing more data. Because cloud providers have several geographical data centers (regions) [14], they use different cost calculation models for additional services (e.g., data migration), as well. Notably, those providers often offer reduced prices, as well, for data migration services between regions.

Apart from the varying pricing, which is based on the regions, pricing models can also differ by storage technologies which are using by cloud vendors. Generally, there are three types of data storing and retrieving technologies which use by cloud providers. Those are *"Hot Data"*, *"Cold Data"*, and *"Cool Data"* [15,16].

In comparison with traditional conventional storages, long-term storage solutions have cheap storage and high traffic prices. Moreover, this type of storage has a minimum storage duration, i.e., Billing Time Unit (BTU) [14]. BTU is independent of API calls charge, and it will not change, whether or not the data was migrated or deleted. Because long-term storages are used mostly for backup purposes, the *BTU* cloud vendors has a charge for the minimum object size: *Billing Storage Unit (BSU)*. *BSU* is a standard pricing methodology for all types of data that have a smaller size than the *BSU* size. *BSU* size will be accepted as a minimum for all types of data, regardless the size of the data is less than that.

While the pricing models used by cloud vendors are based on similar characteristics, there are also significant technological and conceptual differences between them. For instance, Amazon offers storage technologies, like Amazon Simple Storage Service (Amazon S3) and Amazon S3 Standard—Infrequent Access (IA) [17], with 30 days *BTU* and a 128KB *BSU* [18]. For these types of storages, Amazon applies block rate pricing model for the traffic, as well as for the storage. At the same time, Google also applies the block pricing model for their "Google Cloud Storage" but only for outgoing data transfer [19]. At the end of the paper, we provide more information about the pricing of these cloud storage systems.

In our paper, we introduce a new framework for hybrid cloud systems. Our research is mostly focused on optimizing heterogeneous characteristics (throughput, storage cost, etc.) of hybrid cloud storage systems. The main goal of this investigation was to use the advanced performance of Network coding for resource allocation in a flexible manner between public and private cloud storage nodes, to increase upload/download speed, while minimizing the data upload/download and store cost. More explicitly, our approach is to store Network Coded objects in distributed cloud storage entities, e.g., Amazon S3 and Google GCS. In Network Coding, the objects are stored as different random linear combinations of the chunks in each cloud storage. This approach allows us to avoid an overwhelming bookkeeping process when transferring the data between nodes and additional delays from a highly loaded cloud storage. From Figure 1, it can be seen that the original object, which is composed out of several chunks, is coded, and stored in multiple cloud storages. Those chunks use recovery of the original object in the case where the original object will be corrupted, or storage of the object will be replaced with a new one.

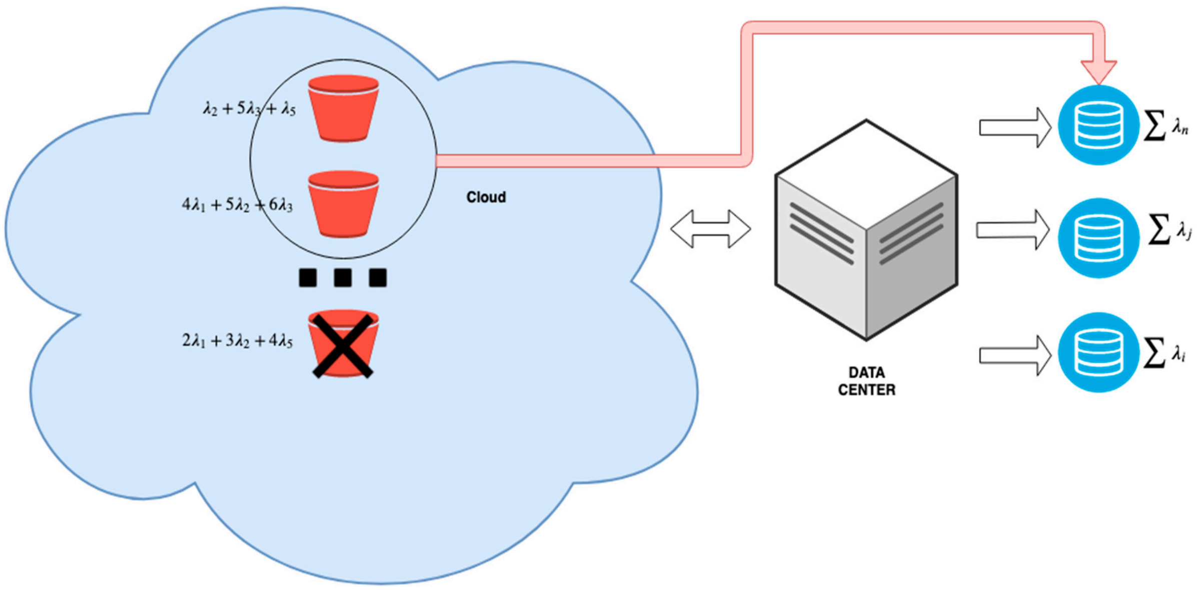
$\lambda_2 + 5\lambda_3 + \lambda_5$
$4\lambda_1 + 5\lambda_2 + 6\lambda_3$
$2\lambda_1 + 3\lambda_2 + 4\lambda_5$
Cloud
DATA CENTER
$\sum \lambda_n$
$\sum \lambda_j$
$\sum \lambda_i$

Figure 1. Architecture of the data chunk placement model.

The results of this paper can be useful for organizations which use hybrid cloud storage systems in the production environment, along with cloud vendors. In upcoming sections, we demonstrate our measurement methods in conjunction with measurement results, using real-life cloud providers, under a variety of scenarios.

3. The Model

In this section, we briefly explain the problem formulation, as well as the data chunk placement model.

3.1. System Characterization

Let us first assume that the system consists of two sets of the storages:

- set $\theta = \{\theta_1, \ldots, \theta_F\}$ to denote the public cloud storages; F is a number of the nodes, and θ_j denotes the jth public node, and
- set $U = \{U_1, \ldots, U_T\}$ to represent the avalaible private storages, where T is number of the private nodes, and U_i denotes the i-th private node, while
- $u = \{u_1, \ldots, u_T\}$ represents private storages sizes.

As was mentioned before, each object consists of the same size fragments (chunks or packets), and those fragments are transferred between the transmission and receiver nodes, i.e., if the object consists of the set of chunks $\Omega = \{\Lambda_{11}, \Lambda_{12}, \ldots, \Lambda_{\mu\varpi}\}$ (where μ represents the cloud vendor, and ϖ is the region of the storage), then the average size of each chunk will be:

$$\Psi_\lambda = \frac{|\Omega|}{\kappa}, \tag{1}$$

where $\kappa$is a number of the fragments, and Ω is the object size. The costs of the data transfer between the nodes are characterized by the following matrix:

$$W = \begin{bmatrix} w_{11} & \ldots & w_{1F} \\ \ldots & \ldots & \ldots \\ w_{T1} & \ldots & w_{TF} \end{bmatrix},$$

where is w_{ab} represents data transfer cost (data transfer rate) between source "a" and receiver "b".

Since some storage nodes may fail, one of the main concerns of system design is reliability. To avoid data loss and ensure system integrity during the building/rebuilding process of the objects, we performed a two-step procedure for ensuring the integrity of the system: in the first step, we check the accessibility of all storage systems, and, if the storage systems are fully available for data transfer, we perform the next step, in which we check the availability of the data.

Even though cloud storage systems were designed to be nearly 100 percent durable, multiple factors (like network failures) can disrupt accessibility of the storage. Furthermore, the encrypted data which we store in cloud storages is vulnerable to attacks [20–22] and can be damaged in multiple ways. If the encrypted data is damaged, we consider the data as unavailable for rebuilding, despite the fact that there is storage available for data transfer. For this reason, we introduced two new terms: the probability of availability (P) and the probability of unavailability (P′). Because the system needs to be stable, we accept that the availability probability of the storages always must be much bigger than unavailability probability: P >> P′. The metric which we use for availability is number of nines [23]. For example, if the availability of the system is 99.9%, then we refer it as three nines. The system with three nines availability is expected to have 8.75 h downtime per year [24]. Thus, it represents system unavailability probability, which is equal to 0.1%. To manage a cloud-to-cloud (public-to-private) data transfer successfully, a certain number of the transfer nodes should be available for rebuilding the object. The overall availability-state of the nodes is defined by an n-dimensional binary vector, where n is the number of the nodes, and $c_\xi = 0$, if the storage is inaccessible; otherwise, $c_\xi = 1$, if is available to transfer data. If we will consider that a cloud node has the availability probability, then, for ideal case, the availability of the overall node will be:

$$P = \sum_{c_1+c_2+\ldots+c_F} \prod_{\xi=1}^{F} p_\xi{}^{c_\xi}. \tag{2}$$

Therefore, the probability of, at most, one node failing is:

$$P = \sum_{c_1+c_2+\ldots+c_F \geq F-1} \prod_{\xi=1}^{F} (1-p_\xi)^{1-c_\xi} p_\xi{}^{c_\xi}. \tag{3}$$

If we will have, at most, N redundant transfer nodes, then the probability of the unavailability of N+1 storages will be:

$$P' = 1 - \sum_{c_1+c_2+\ldots+c_F \geq F-N} \prod_{\xi=1}^{F} (1-p_\xi)^{1-c_\xi} p_\xi{}^{c_\xi}, \tag{4}$$

as

$$P' = 1 - \sum_{\xi=0}^{N} \begin{pmatrix} F \\ \xi \end{pmatrix} \cdot (1-p)^{\xi} p^{F-\xi}, \tag{5}$$

where $P' < 0.001$.

With summarizing above conditions, we define the reliability of the system as below:

Definition 1. *(Unavailability Probability): Let P′ denote the unavailability probability of the system. In order to keep the probability of data loss under a threshold during the rebuilding and transferring objects, unavailability probability of the system P′ should be less than 0.001* [14].

- The "Quality of Service" (QoS) criterion on reliable file upload implies that the number of the fragments stored in the transfer nodes must be sufficient for rebuilding the object. Let's assume that we have $Y = \sum_{\tau=1}^{v} y_\tau$ fragments for yielding, $Y \geq \Omega$. For the specific cases where we have N redundant node with probability P′, the QoS criteria will be $P(\sum_{\tau=1}^{v} y_\tau \geq \Omega) > 1 - P'$.

3.2. *Optimal Uploading Criteria*

The optimal uploading criteria is formulated by the following steps:

- The chunks transfer from the public node to the receiver node is represented by a binary matrix X, where the elements are $\mathrm{x_{ijg}}= \{0, 1\}$ being transferred from node *i* to substorage *g* of node *j*.
- The overall size of the chunks, which were transferred to receiver nodes, should not exceed the storage size. Namely, the data reliability criteria of the QoS standard should not be violated.

3.3. *Formulation of the Optimal Upload Strategy*

To meet the criteria described above, we designed an optimal uploading scheme. We can formalize our optimal uploading problem as searching for optimal W_{opt} as follows:

$$W_{\text{opt}}\colon \ \min\sum_{i=1}^{T}\sum_{j=1}^{F}\sum_{g=1}^{k} w_{ij}x_{ijg}\Psi_{\Lambda_{ig}}, \tag{6}$$

under the conditions of

$$\sum_{g=1}^{k} x_{ijg}\Psi_{\Lambda_{ig}} \le u_{bj}, k \le F, b = 1, .., \mathrm{d}, j = 1, .., \mathrm{T}, i= 1,.., \mathrm{F}, \tag{7}$$

$$\sum_{i=1}^{T}\sum_{j=1}^{F}\sum_{g=1}^{k} \gamma_{ij}\Psi_{\Lambda_{ig}} \le \sum_{j=1}^{F} u_{bj}\ b = 1, .., \mathrm{d}, \tag{8}$$

where γ_{ijg} represents binary transfer matrix between nodes, and u_{bj} represents the overall size of the sub-storages in the single vendor.

$$\mathrm{P}(\sum_{\tau=1}^{\mathrm{v}} \mathrm{y}_{\tau} \ge \Omega) \ge \mathrm{P}'. \tag{9}$$

The main purpose of constraints (7) and (8) is to avoid exceeding the size of the sub storage and node (where (7) was used for sub storage and (8) for the node) during the data transfer.

4. Heuristics for Finding an Optimal Data Chunk Placement

In this section, we propose an algorithm for finding the optimal data chunk placement. To obtain this, we should minimize the cost of the object rebuilding, while providing full-scale search.

4.1. *Data Chunk Change Criteria*

Let us assume that we have an object which consists of the chunk vector $\Lambda' = (\Lambda'_1, \Lambda'_2, \ldots, \Lambda'_{\mathrm{n}-1}, \Lambda'_{\mathrm{n}})$. To meet reliability criterion, let us accept that the availability probability of all necessary storages for data transfer is $\mathrm{p}(\theta')$ and $\mathrm{p}(\theta') > \mathrm{p}'(\theta')$. If the probability of transferring at least Y chunks ($\mathrm{p}(Y)$) is to be increased in order to fulfill the second reliability criteria, then there is the only way to do this: by transferring chunk from the storage θ' (where is $\theta' \in \{\theta_1, \ldots, \theta_{\mathrm{F}}\}$) to the storage U' (where is $\mathrm{U}' \in \{\mathrm{U}_1, \ldots, \mathrm{U}_{\mathrm{T}}\}$). Therefore, when the chunk is transferring to receiver storage, the third reliability criteria should be obtained, which means $\sum_{\mathrm{i}=1} \Lambda_{\mathrm{i}} \le \mathrm{u}'$ (where is $\mathrm{u}' \in \{\mathrm{u}_1, \ldots, \mathrm{u}_{\mathrm{T}}\}$).

4.2. *Moving Chunks from the Transmitter to Receiver Nodes*

Let us suppose that the chunk transfer combination specified as the vector $\mathrm{Y} = (\Lambda''_1, \Lambda''_2, \ldots, \Lambda''_{\mathrm{n}-1}, \Lambda''_{\mathrm{n}})$ and the object itself is $\Omega = (\Lambda_{11}, \Lambda_{12}, \ldots, \Lambda_{\delta \mathrm{n}})$. If the single chunk was transferred from Y to Ω, then both vectors change as $\mathrm{Y} = (\Lambda_1, \Lambda_2, \ldots, \Lambda_{\mathrm{n}-1})$

and $\Omega = (\lambda'_{11}, \lambda'_{12}, \ldots, \lambda'_{\beta n}, \Lambda_n)$, where λ' represents existing chunks in the storage. The reliability has been increased by this state change as an additional chunk is available for rebuilding object in the case where the object will be corrupted or totally lost. However, in this case, the cost object rebuilding is also increasing, but the transformation is defined in such a way that the required minimal cost of the chunk rebuilding is defined as an optimal for the current type operation. In the next section, we describe this process more briefly.

4.3. Data Chunk Placement Design Algorithm

To provide full-scale search, the availability probability of the storages was checked first. If the availability is $p(\theta') > p'(\theta')$, then set the initial cost of the transfer $W_i = 0$.

Start to search for the chunk to rebuild the object. If the chunk Λ_m was transferred, then the overall transfer cost changes. In this case, the state matrix update for the given transfer storage updates from "0" to "1".

Check the conditions (8), (9), and (10); if the conditions are fulfilled, then update the overall minimum cost for the data transfer, $W_i = W_i'$. If, in the next iteration, the new overall cost $W_i'' < W_i$, then $W_i = W_i''$; otherwise, the old one keeps continuing to search. There are three conditions for the termination of the algorithm:

- Availability probability of the data transfer storages decreases to less than 0.999. This case is violating the first reliability condition of the system.
- Probability of the successfully rebuilding object became less than the probability of the unsuccessfully rebuild $P(Y') < p'$.
- There is no more transfer storage to provide search.

5. Performance Results

The proposed algorithm in this paper was evaluated with two objectives:

- find the minimum data transferring cost for the ideal case, when all storages are available for data transfer and rebuilding;
- find the minimum data transfer cost, when there is one or multiple storages are redundant in the system.

To reach the optimal solution, a full-scale search was performed. However, because of the large processing time and resource consumption required to process a large amount of data, the optimal solution search was only run on a small data set. Our aim was to test the algorithm on real-life scenarios; thus, we compared our results with our previous model OUS [25], as well as with scenarios when the single cloud vendor storage (either GCP or AWS) was used for data uploading. The common approach between these three models is not having a recovery strategy for data uploading from the breakpoint. Moreover, while OUS can transfer from multi-cloud vendors, AWS and GCP do not have this functionality. OUS also distributed data over the nodes in the network, yet it did not reconstruct the whole data in the network coding style, due nature of the model. This was a reason why we decided to test our model for different cases and compare performance with all these three data uploading models.

For information collecting purposes, we used existing metric systems of the cloud providers.

5.1. Testbed Setup and Metrics

For setting up testbed and showcasing the previously presented analytic results, we emphasized existing research studies [26,27] in the field. For encoding and decoding the data object, we use KODO-Python: Python binding for KODO Network Coding library [28] which was written in C++ language.

For testing purposes, we took multiple random files with the size of 16 MB, and we split them into the chunks with the size of 512 KB. Moreover, for the current model, we used generation size 32 and the coefficient which was chosen from the finite field with a size of GF(2^8) [29].

The model was simulated in the environment, where the server has 50 GB of RAM and 12 CPU with 6 cores per Socket. In addition to this, we also created a simulated platform of a hybrid cloud multi-node system which consists of 8 storages (Google Nearline and S3 Standard-Infrequent Access), running in different regions, as well as 2 local storages, which was replicated receiver nodes. Receiver storage size was artificially limited maximum to 100 MB for testing purposes.

For our experiments, we used cloud vendors to cool data pricing strategy. "Cool Data" has a lower storage cost compared with its predecessors. However, it has extra operation fees per request and retrieval charge for every GB of downloaded data. As was presented above, the early deletion fee is another outlying item for "Cool Data" storage tier. No matter that the data was stored during the period or removed before the last day of the BTU, the minimum storing span will always go up. As an example, if the 1-GB data was stored on day 1 and removed on day 9 in Google Cloud Storage, the charge will be:

- $0.01 (0.01$ is approximate price for GCP storage) * 1 GB * 9/30 = $3 for storing the data from day 1 to 9;
- $0.01 * 1 GB * 21/30 = $7 for storing the data from day 9 to 21, 10$ in total.

Both of the cloud vendors which were chosen for the testing purpose have a free tier for a certain amount of the REST API calls and data retrieval operation with multiple restrictions, such as after 1 GB and 1000 requests to "S3 Standard-Infrequent Access" storages, where Amazon charges 0.0001–0.1$ per 1000 requests, depending on the type of the request [30]. The price also varies based on the regions of the cloud storage. For our experiment, we use the same pricing scheme for per 10 requests (GET request only) and 1 MB of the data (due to a size small size of the data which we used for testing).

Further measurement was done using official Amazon "S3 Transfer Acceleration" [29] and Google "Google Cloud's Operations Suite" [31] tools. For storage metrics, we used the "CloudWatch" [32] monitoring service. All metrics information which was used for analyzing the performance of the algorithm is summarized in Tables 1 and 2.

Table 1. General metric parameters.

Parameter Name	Value
Number of the Transmitter Storage	8
Number of the Receiver Storage	2
Number of the Chunks per file	32
Storage Capacity Range (KB)	[4096, ... , 102,400]
Chunk size (KB)	512
Locations of the receiver Storages	[Baku, Budapest]
AWS Transmitter Storages Location	[Seoul, Cape Town, Frankfurt, Oregon]
GCP Transmitter Storages Location	[Zurich, Mumbai, Osaka, Montreal]

Table 2. Simulating parameters (measured in real time).

Region Name	GET Request	From Location	Data Transfer OUT/GB	Storing Cost
AWS Asia Pacific (Seoul)	0.00035 $	Baku	0.126 $	0.025 $
AWS Europe (Frankfurt)	0.00043 $	Budapest	0.09 $	0.0245 $
AWS US West (Oregon)	0.0004 $	Budapest	0.09 $	0.023 $
AWS Africa (Cape Town)	0.0004 $	Baku	0.154 $	0.0274 $
GCP Europe-West6 (Zurich)	0.01 $	Budapest	0.010 $	0.014 $
GCP Asia-South1 (Mumbai)	0.01 $	Budapest	0.010 $	0.016 $
GCP Asia-Northeast2 (Osaka)	0.01 $	Baku	0.012 $	0.016 $
GCPNAmerica-NorthEast1 (Montreal)	0.01 $	Baku	0.009 $	0.013 $

5.2. Testing Results

To test the performance of the algorithm, we took 6 files with the same size and split them to 192 chunks with size 512 KB, as described in Table 1. As the initial step, we distributed chunks over the transmitter storages because OUS and traditional cloud storages have no Network Coding functionality.

In the first experiment, we wanted to determine the best data uploading strategy based on the varying costs of the vendors for different regions and compare our result with the single-vendor solution (either AWS or GCP) and OUS performance. In this experiment, we assumed that there was no redundant storage in the system, and all storages were fully available for data transfer. Besides, we accepted that each request is returning a 512 KB single chunk, and all requests go parallel.

For the first case, both of the algorithms, OUS and DCP, performed well, with a minor difference, compared to single cloud data transfer solutions. In contrast to manual data uploading/downloading, where the cost of the data transfer cannot always be tracked, OUS and DCP have better performance because both algorithms always seek the minimum cost. Figure 2 describes how the minimum cost increased in four different environments during object rebuilding.

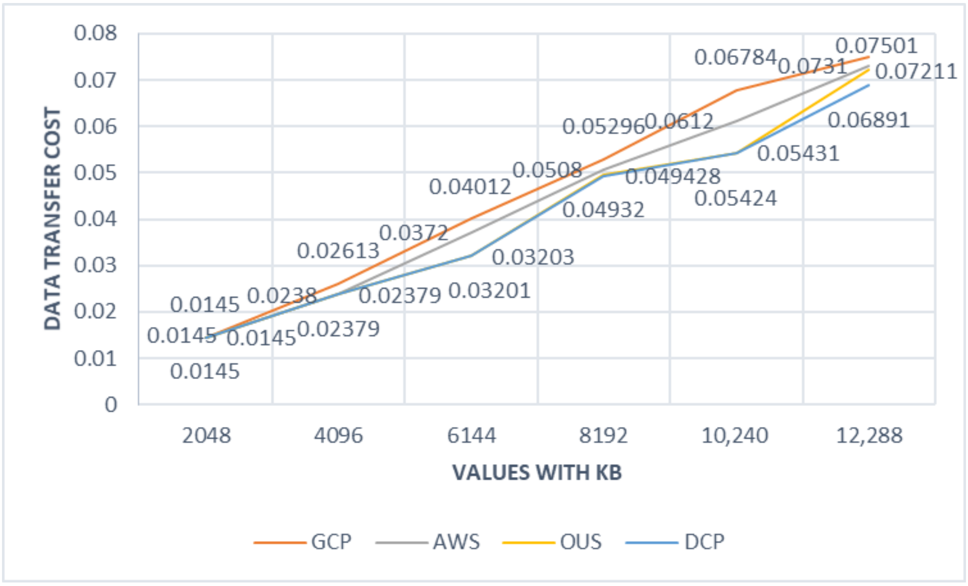

Figure 2. Data transfer cost change without loss in the system.

In the second phase of the test, we followed the same approach as in the first test but with some changes. Normally, all cloud storages have an availability probability of 99.99%, which we cannot change. For this reason, we created damaged data sets. This type of data set cannot be used for object rebuilding because of the lack of bits.

After testing both of algorithms, we admitted that DCP had a better performance compared to OUS and single cloud vendor solutions. Even though, in OUS, we also calculated overall cost with using minimum data transfer cost, OUS has no option for rebuilding an object from broken point (in this case where OUS starts the object rebuilding process from the beginning again), thus increasing overall cost data object rebuilding process. The second test actually shows the benefits of using Network Coding versus traditional data transfer strategies to cloud storages. Figure 3 describes the second test cases for the evaluation of the algorithm performance.

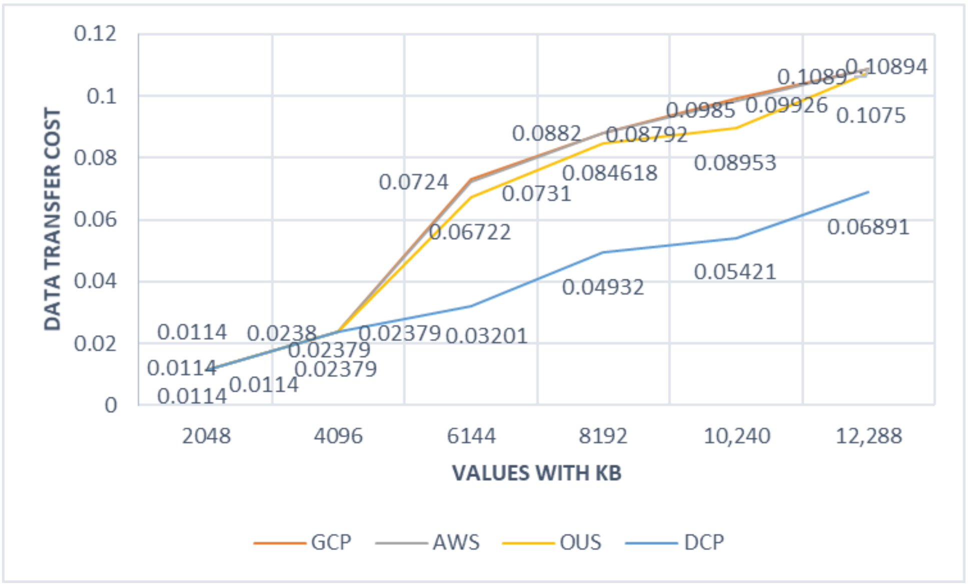

Figure 3. Data transfer cost change with redundant node.

From both of the experiments, it can be seen that overall performance of the DCP is much better than OUS and single vendor data uploading strategies. Furthermore, because of the Network Coding performance, DCP has significant benefits over the mentioned technics.

5.3. Proposed Model Implementation

DCP can be implemented as part of both interim data transfer between public and private cloud storage clustering, as well as part of the multi-part upload to cloud storage systems from a single location. For the cases where the model was used for interim data transfer, the performance shared storage system will be improved significantly due to the provided near-optimal solution. Because the model is taking care of the nodes' unavailability, clustered storage systems issues, like redundancy, will also be minimized.

As part of the multi-part uploading process, it is extremely essential to have a minimum upload cost for the POST requests. DCP can help to achieve it, with search performance, where is the model looks for real-time cost-optimal storage from the provided location. This can help to optimize the SLA agreement between provider and client.

6. Conclusions

In this paper, we introduced a novel approach for distributed storage uploading strategy to cloud storages. The proposed optimization method is capable of rebuilding any object with a near-minimal transfer cost. The algorithm can especially be useful for big organizations, which are seeking a way for the integration of private cloud storages to the public cloud with less secure concern.

With using Network Coding technics, the data chunk optimization algorithm covers issues related to network lost during the data transferring. This functionality is essential for organizations, where non-stop data transfer happening between internal and cloud storage nodes.

As future work, we are going to investigate the performance of the algorithm on different real-world tasks, in real-time. Moreover, we will analyze the performance of the different optimization algorithms on our model and will compare the existing result with them. In addition to this, we will try to integrate the uploading strategy to open source libraries which are using for data uploading to cloud storages.

In addition, another challenge is cloud vendors' pricing models, which are quite changeable and based on the multiple factors (such as data storing prices decrease with increasing the size/amount of the data in the storage). As a part of further research, we will also work on improving our model for these factors.

Besides the optimization of the data uploading process, we will also extend our algorithm, for the data streaming process, where we will introduce an extended algorithm for incremental data.

Author Contributions: Conceptualization, J.L. and A.Y.; methodology, J.L. and A.Y.; software, J.L. and A.Y.; validation, J.L. and A.Y.; formal analysis, A.Y.; investigation, A.Y.; resources, J.L.; writing—original draft preparation, A.Y.; writing—review and editing, J.L.; visualization, A.Y.; supervision, J.L.; project administration, J.L.; funding acquisition, J.L.; All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Artificial Intelligence National Laboratory.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: All measurements are available publicly. There is no specific data report.

Acknowledgments: The research presented in this paper, carried out by the Budapest University of Technology and Economics was supported by the Ministry of Innovation and the National Research, Development and Innovation Office within the framework of the Artificial Intelligence National Laboratory Programme.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Waibel, P.; Matt, J.; Hochreiner, C.; Skarlat, O.; Hans, R.; Schulte, S. Cost-optimized redundant data storage in the cloud. *Serv. Oriented Comput. Appl.* **2017**, *11*, 411–426. [CrossRef]
2. Dimakis, A.G.; Godfrey, P.B.; Wainwright, M.J.; Ramchandran, K. Network Coding for Distributed Storage Systems. In Proceedings of the IEEE INFOCOM 2007—26th IEEE International Conference on Computer Communications, Anchorage, AK, USA, 6–12 May 2007.
3. Jereb, L. Network Reliability: Models, Measures and Analysis. In Proceedings of the 6th IFIP Workshop on Performance Modelling and Evaluation of ATM Networks, Tutorial Papers, Ilkley, UK, 21–23 July 1998.
4. Balaji, S.B.; Krishnan, M.N.; Vajha, M.; Ramkumar, V.; Sasidharan, B.; Kumar, P.V. Erasure coding for distributed storage: An overview. *Sci. China Inf. Sci.* **2018**, *61*, 100301. [CrossRef]
5. Dimakis, A.G.; Ramchandran, K. Network Coding for Distributed Storage in Wireless Networks. In *Networked Sensing Information and Control*; Saligrama, V., Ed.; Springer: Boston, MA, USA, 2008.
6. Jammal, M.; Kanso, A.; Heidari, P.; Shami, A. Availability Analysis of Cloud Deployed Applications. In Proceedings of the 2016 IEEE International Conference on Cloud Engineering (IC2E), Berlin, Germany, 4–8 April 2016; pp. 234–235.
7. Abu-Libdeh, H.; Princehouse, L.; Weatherspoon, H. RACS: A case for cloud storage diversity. Cloud Computing. In Proceedings of the 1st ACM symposium on Cloud computing—SoCC '10, Indianapolis, IN, USA, 10–11 June 2010.
8. Waibel, P.; Hochreiner, C.; Schulte, S. Cost-Efficient Data Redundancy in the Cloud. In Proceedings of the 2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA), Macau, China, 4–6 November 2016; pp. 1–9.
9. Chen, H.C.H.; Hu, Y.; Lee, P.P.C.; Tang, Y. NCCloud: A Network-Coding-Based Storage System in a Cloud-of-Clouds. *IEEE Trans. Comput.* **2013**, *63*, 31–44. [CrossRef]
10. Al-Roomi, M.; Al-Ebrahim, S.; Buqrais, S.; Ahmad, I. Cloud Computing Pricing Models: A Survey. *Int. J. Grid Distrib. Comput.* **2013**, *6*, 93–106. [CrossRef]
11. Tudoran, R.; Costan, A.; Antoniu, G. Transfer as a Service: Towards a Cost-Effective Model for Multi-site Cloud Data Management. In Proceedings of the 2014 IEEE 33rd International Symposium on Reliable Distributed Systems, Nara, Japan, 6–9 October 2014; pp. 51–56.
12. Graupner, H.; Torkura, K.; Berger, P.; Meinel, C.; Schnjakin, M. Secure access control for multi-cloud resources. In Proceedings of the 2015 IEEE 40th Local Computer Networks Conference Workshops (LCN Workshops), Clearwater Beach, FL, USA, 26–29 October 2015; pp. 722–729.
13. Zhang, Q.; Li, S.; Li, Z.; Xing, Y.; Yang, Z.; Dai, Y. CHARM: A Cost-Efficient Multi-Cloud Data Hosting Scheme with High Availability. *IEEE Trans. Cloud Comput.* **2015**, *3*, 372–386. [CrossRef]
14. Frîncu, M.E.; Genaud, S.; Gossa, J. Comparing Provisioning and Scheduling Strategies for Workflows on Clouds. In Proceedings of the 2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum, Cambridge, MA, USA, 20–24 May 2013.

15. Irie, R.; Murata, S.; Hsu, Y.-F.; Matsuoka, M. A Novel Automated Tiered Storage Architecture for Achieving Both Cost Saving and QoE. In Proceedings of the 2018 IEEE 8th International Symposium on Cloud and Service Computing (SC2), Paris, France, 18–21 November 2018.
16. Hsu, Y.-F.; Irie, R.; Murata, S.; Matsuoka, M. A Novel Automated Cloud Storage Tiering System through Hot-Cold Data Classification. In Proceedings of the 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), San Francisco, CA, USA, 2–7 July 2018; pp. 492–499.
17. Amazon S3 Storage Classes. Available online: https://aws.amazon.com/s3/storage-classes/ (accessed on 26 July 2020).
18. Storage Class for Automatically Optimizing Frequently and Infrequently Accessed Objects. Available online: https://docs.aws.amazon.com/AmazonS3/latest/dev/storage-class-intro.html (accessed on 26 July 2020).
19. Google Cloud Network Pricing. Available online: https://cloud.google.com/compute/network-pricing (accessed on 26 July 2020).
20. Fan, C.-I.; Huang, S.-Y.; Hsu, W.-C. Encrypted Data Deduplication in Cloud Storage. In Proceedings of the 2015 10th Asia Joint Conference on Information Security, Kaohsiung City, Taiwan, 24–26 May 2015; pp. 18–25.
21. Zhang, Y.; Xu, C.; Cheng, N.; Shen, X. Secure Encrypted Data Deduplication for Cloud Storage against Compromised Key Servers. In Proceedings of the 2019 IEEE Global Communications Conference (GLOBECOM), Big Island, HI, USA, 9–13 December 2019; pp. 1–6.
22. Gochhayat, S.P.; Bandara, E.; Shetty, S.; Foytik, P. Yugala: Blockchain Based Encrypted Cloud Storage for IoT Data. In Proceedings of the 2019 IEEE International Conference on Blockchain (Blockchain), Atlanta, GA, USA, 14–17 July 2019; pp. 483–489.
23. Chang, C.-W.; Liu, P.; Wu, J.-J. Probability-Based Cloud Storage Providers Selection Algorithms with Maximum Availability. In Proceedings of the 2012 41st International Conference on Parallel Processing; Institute of Electrical and Electronics Engineers (IEEE), Pittsburgh, PA, USA, 10–13 September 2012; pp. 199–208.
24. Mansouri, Y.; Toosi, A.N.; Buyya, R. Brokering Algorithms for Optimizing the Availability and Cost of Cloud Storage Services. In Proceedings of the 2013 IEEE 5th International Conference on Cloud Computing Technology and Science, Bristol, UK, 2–5 December 2013; Volume 1, pp. 581–589.
25. Yolchuyev, A. A Novel Approach for Optimal Data Uploading to the Distributed Cloud Storage Systems. In Proceedings of the 2019 Big Data, Knowledge and Control Systems Engineering (BdKCSE), Sofia, Bulgaria, 21–22 November 2019; pp. 1–6.
26. Tan, P.; Chen, Y.; Li, C. A secure regenerating code for the fault-tolerant of distributed networked storage. In Proceedings of the 2013 IEEE 4th International Conference on Software Engineering and Service Science, Beijing, China, 23–25 May 2013; pp. 507–510.
27. Abdrashitov, V.; Medard, M. Durable network coded distributed storage. In Proceedings of the 2015 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton), Monticello, IL, USA, 30 September–2 October 2015; pp. 851–856.
28. Pedersen, M.V.; Heide, J.; Fitzek, F.H.P. Kodo: An Open and Research Oriented Network Coding Library. In *NETWORKING 2011 Workshops*; NETWORKING 2011. Lecture Notes in Computer Science; Casares-Giner, V., Manzoni, P., Pont, A., Eds.; Springer: Berlin/Heidelberg, Germany, 2011.
29. Amazon S3 Transfer Acceleration Speed Comparison. Available online: https://s3-accelerate-speedtest.s3-accelerate.amazonaws.com/en/accelerate-speed-comparsion.html (accessed on 26 July 2020).
30. How to Switch Between Classes in Google Storage Service. Available online: https://cloud.netapp.com/blog/google-storage-service-how-to-switch-google-cloud-storage-class (accessed on 26 July 2020).
31. Google Cloud's Operations Suite. Available online: https://cloud.google.com/products/operations (accessed on 26 July 2020).
32. Amazon CloudWatch Pricing. Available online: https://aws.amazon.com/cloudwatch/pricing/ (accessed on 26 July 2020).