Efficient Data Access in Hybrid Cloud Storage

Islam Samy
ECE, Univ. of Arizona,
Tucson, AZ 85721, USA.
Email: islamsamy@email.arizona.edu

O. Ozan Koyluoglu
EECS, UC Berkeley,
Berkeley, CA 94720, USA.
Email: ozan.koyluoglu@berkeley.edu

Ankit Singh Rawat
RLE, MIT,
Cambridge, MA 02141, USA.
Email: asrawat@mit.edu

***Abstract*—Hybrid cloud is a widely adopted framework where on-premise storage and/or compute resources are combined with public cloud system. This paper explores the storage aspect of this framework, which requires designing coding schemes that are aware of both local and global components of the available storage space. The coding schemes should provide efficient repair mechanisms for the data stored on the public cloud (global storage space) and utilize the local storage space to facilitate seamless access to the overall information stored on the hybrid cloud storage. This paper presents a mathematical model for hybrid cloud storage which takes all these requirements into account. The paper then extends the information flow graph approach to characterize the fundamental limits on access bandwidth of the system, i.e., the amount of data downloaded from the public cloud during the data reconstruction process. This paper also presents several explicit coding schemes that utilize the available local storage space to attain the fundamental limit on the access bandwidth. The setup where multiple clients with varying local storage spaces are supported by a single global storage space is also addressed.**

I. Introduction

Cloud computing is one of the most pervasive technology paradigms in the recent past that has transformed the ways with which various businesses operate. The cloud systems host basic IT resources, including storage and compute, that are made available to businesses on their demand with varying degrees of control. This frees these institutions from the burden of managing most of the IT resources needed for their successful operations; consequently, realizing improved efficiency with significant reductions in their day-to-day operating costs.

However, various practical and policy concerns prevent complete migration to *public cloud* systems, where data is stored and processed in the data centers hosted by a third-party vendor. For example, the latency of data access from a public cloud is unavoidably higher than that from the storage locally available at the firm. This gap in latency can significantly increase during a catastrophic event which might hamper the critical operations [1]. Additionally, businesses may not want to host sensitive and valuable information on public clouds which are subject to various covert attacks [2], [3], where unauthorized agents may gain access to the valuable information through shared hardware resources.

These concerns motivate the adoption of *hybrid cloud* systems, where local IT resources of a business are combined with the remote IT resources provided by a third parity vendor in a single environment (cf. Figure 1). This enables the business to locally host the sensitive information along with the mission critical data processing. The ability to access the remote resources within a single environment allows the businesses to employ *cloud-bursting*, where they can delegate the processing tasks to remote resources during the spikes in the demand for compute resources. Furthermore, businesses can meet their growing storage needs by storing the cold or warm data on the remotely available global storage space.

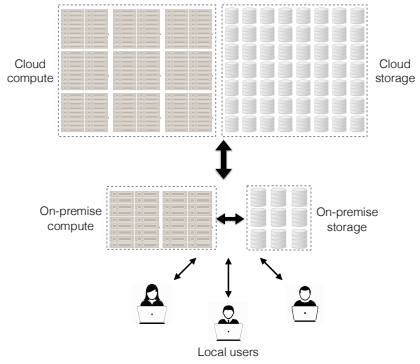

Fig. 1: Generic setup of a hybrid cloud system. Such a cloud system combines the limited compute and storage resources available locally at a business with the large scale compute and storage resources publicly hosted by a third party vendor.

In this paper, we focus on a coding theoretic treatment of the storage aspect of the hybrid cloud systems. Recently, the problem of designing reliable and efficient cloud storage (a.k.a. distributed storage systems) has received a great deal of attention from the research community (see e.g., [4]–[10] and references therein). In particular, distributed nature of cloud storage presents challenges that are unique to cloud storage systems. So called *node repair* problem is one such challenge, which necessitates designing coding schemes that allow for resource efficient mechanisms to reconstruct the small portion of stored information [5]. The node repair problem is motivated by the frequent scenarios in real-life storage systems where a small number of storage nodes in the system are lost or temporarily inaccessible. The progress in addressing the issue of node repair has led to many novel code constructions (e.g., [6], [11]–[18]) and application of the obtained codes in real-life systems [7]–[10].

This work is supported in part by NSF awards CCF-1748585 and CNS-1748692.

However, most of the existing literature does not address the design of hybrid cloud storage systems as it primarily studies a setup which is closer to the public cloud storage. The existing work does not focus on the inherent asymmetry between local and global storages spaces, which is present in a hybrid cloud storage system. This paper aims to address this issue by developing a theoretical framework that takes the specific requirements of hybrid cloud storage into account. The framework ensures that the information is reliably stored on a hybrid cloud storage system so that it is possible to perform repair-bandwidth efficient node repair in the global storage space. This ensures seamless low cost operation and availability of the remote storage resources. Additionally, the framework exploits the local storage space to improve the efficiency of the data access in a hybrid cloud storage system. In particular, a data access request from an end user can be served by combining the information stored on a sufficient number of storage nodes in the global storage space with the content of the local storage space. This leads to the reduced utilization of various system resources, including the bandwidth between the remote storage nodes and the end user.

We first generalize the information flow graph approach from [5] to handle the asymmetry between local and global storage spaces. The generalized approach helps us obtain a lower bound on the access-bandwidth of a hybrid cloud storage system, which refers to the amount of data downloaded from the global storage nodes during a data access. We also employ information flow graph approach to study the setting where multiple local storage spaces with varying sizes interact with a global storage space. We then utilize existing code constructions for repair-bandwidth efficient codes to obtain coding schemes that ensure small access-bandwidth. The obtained schemes also attain the bound on access bandwidth for a large range of system parameters. Here, we note that our proposed schemes for hybrid cloud storage utilize the entire storage space comprising of local and global resources as a single environment. This is consistent with the approach of having a single namespace across the entire storage space in real-life hybrid cloud systems [19].

A. Related work

In [20], Maddah-Ali and Niesen study a caching setup where local storage close to end-users is utilized to reduce real-time communication from a content server to the end-users irrespective of their request pattern. There are multiple key differences between the caching setup from [20] and the hybrid storage model considered here. One such difference is that Maddah-Ali and Niesen only assume a centralized server and do not focus on the distributed storage setup at the storage side. Therefore, they do not take the server side data management into account which is one of the main objectives in our work. Another difference is the mode of operation. As compared to the caching setup, the consumption of the stored content in a hybrid storage model is much more restricted as the business deploying the hybrid storage has reasonable control over both the stored content and the request pattern. We refer the reader to [21] and the references therein for the large amount of follow-up work on [20].

In [22], Luo et al. explore a caching setup [20] with coded distributed storage at server side. They focus on the setting where coding scheme with a small number of parities are employed by the distributed storage system serving the users with local caches. Similarly, in [23], Aggarwal et al. study the cache optimization issues in the presence of coded distributed storage system. However, both of these works do not address the management of the distributed storage, namely node repairs. In this paper, in addition to focusing on the node repairs, we also explore the trade-off between local storage and access bandwidth. We note that, [24] considered the hybrid cloud system with security constraints, where the focus was on how one can benefit from the local storage by storing random symbols in order to provide security against eavesdroppers.

Organization. The rest of the paper is organized as follows. Section II presents the system model for hybrid storage studied in this paper. We also give a background on the repair-bandwidth efficient node repair in distributed storage systems [5] and the product-matrix code construction by Rashmi et al. [11]. Section III presents the min-cut analysis on the information flow graph associated with our hybrid storage model and explores the extreme points of the obtained storage vs. repair-bandwidth trade-off. Section IV and V utilize the product-matrix framework to obtain coding schemes that exploit local storage to make data access efficient at the two extreme points of the trade-off, respectively. Section VI concludes the paper by identifying a few interesting directions for future research.

II. Background

A. System Model

We consider a hybrid cloud storage system which utilizes a network of n storage nodes as the global storage space. Each of these nodes has capacity of storing α symbols from a finite field $\mathbb{F}$. Additionally, the system has a local storage at its disposal that can store L symbols from $\mathbb{F}$. Throughout this paper, we treat the local storage as a single centralized unit which the client can access with very small latency. Let $\mathbf{f}$ be a file that needs to be stored on the hybrid cloud storage. We assume that the file $\mathbf{f}$ comprises $\mathcal{M}$ symbols from $\mathbb{F}$. The file $\mathbf{f}$ is first encoded into a codeword containing $n\alpha + L$ symbols of $\mathbb{F}$, which are then stored on the system. Note the total storage capacity of the system is $n\alpha + L$. We assume that the underlying coding scheme is such that the content of any k out of n storage nodes is sufficient to reconstruct all the symbols stored on the global space.

Since node failures are inevitable in a large scale storage network, we require the ability to perform repair of the storage nodes in the global storage space. In the event of a single node failure, the repair process involves replacing the failed node by a newcomer node, which regenerates the content of the failed node by downloading β symbols of $\mathbb{F}$ from d helper nodes. We allow the newcomer to contact any

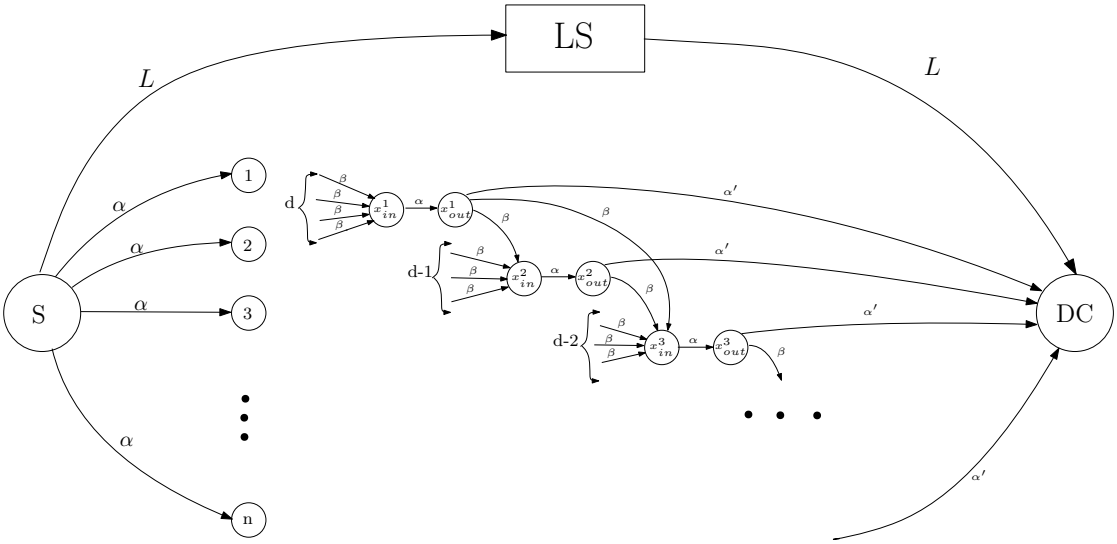

Fig. 2: Information flow graph representation of a hybrid cloud storage system.

set of d out of $n-1$ remaining intact nodes. Note that the total repair-bandwidth associated with the repair of a single node failure is $\gamma = d\beta$.

We use η to denote *access bandwidth* – the amount of information dowloaded by the client or *data collector* to reconstruct the stored file $\mathbf{f}$. We assume that the data access process involves contacting k out of n storage nodes. Therefore, the access bandwidth is clearly upper bounded as $\eta \leq k\alpha$. We aim to exploit local storage space to reduce the access bandwidth η. Towards this, we require the client to download $\alpha' < \alpha$ symbols from each of the k contacted nodes. The $k\alpha'$ downloaded symbols are then combined with the locally stored L symbols in order to reconstruct the file $\mathbf{f}$. This translates to the access bandwidth $\eta = k\alpha' \leq k\alpha$.

B. Information flow graph

We now briefly describe the information flow graph representation of a distributed storage system. The min-cut analysis on these graphs was first utilized in [5] to explore the storage vs. repair-bandwidth trade-off in a distributed storage system. Here, we modify the information flow graph to model the local storage aspect of a hybrid cloud storage (cf. Figure 2). The modified information flow graph contains four kinds of nodes:

- **Source node** S**:** This node represents the collection of $\mathcal{M}$ symbols of the file $\mathbf{f}$ that needs to be stored in the distributed storage system. The source node is connected to n storage nodes with edges of capacity α.
- **Storage nodes** $\{x^i\}$**:** Each storage node comprises two sub-nodes: (i) input node x^i_{in} and (ii) output node x^i_{out}. The input nodes represent the symbols downloaded by each node, while output nodes represent the α symbols stored in each storage node. The two sub-nodes associated with a storage node x^i are connected with an edge of capacity α. In the repair process, the input node of any newcomer node x^j_{in} is connected to the output node of the d helping nodes with edges of capacity β.
- **Data collector node** DC**:** This node represents the client which needs to reconstruct the stored file. The DC is connected to a set of k nodes with edges of capacity α'.
- **Local storage node** LS**:** This node is connected to both the source node and the data collector node with edges of capacity L, that represents the size of the local storage.

Remark 1. *The modified information flow graph representation in Figure 2 differs from the traditional information flow graph [5] in two ways. First, the local storage node is not present in the traditional flow graph as the local storage aspect is not explored in [5]. Besides this, in the modified flow graphs, the data collector node is connected to k storage nodes with edges of capacity α'. In [5], the similar edges are assumed to have infinite capacity. In fact, the presence of these edges with finite capacity allows us to characterize the access bandwidth η for a hybrid cloud storage system.*

Based on the min-cut analysis of the information flow graph in [5], Dimakis et al. obtain the following bound on the size of the stored file in terms of per-node storage α and repair-bandwidth $\gamma = d\beta$.

$$\mathcal{M} \leq \sum_{i=0}^{k-1} \min\big((d-i)\beta, \alpha\big). \tag{1}$$

Recall that the setup considered in [5] has $\alpha' = \alpha$ and $L = 0$. For a given file size $\mathcal{M}$, the bound in (1) clearly establishes a trade-off between per-node storage and repair-bandwidth. In particular, the two extreme points of this trade-off are referred to as the minimum repair-bandwidth regeneration (MBR) point and the minimum storage regeneration (MSR) point, respectively. The per-node storage and repair-bandwidth at

these two extreme points take the following values.

$$\text{MBR:}\quad (\alpha,\beta)=\left(\frac{2\mathcal{M}\ d}{2kd-k^2+k},\frac{2\mathcal{M}}{2kd-k^2+k}\right), \tag{2}$$

$$\text{MSR:}\quad (\alpha,\beta)=\left(\frac{\mathcal{M}}{k},\frac{\mathcal{M}}{k(d-k+1)}\right). \tag{3}$$

The codes that attain these two points are referred to as the MBR codes and the MSR codes, respectively. There are multiple explicit constructions of these codes known in the literature. In [11], Rashmi et al. present product-matrix framework to obtain constructions of both MBR codes and (low rate) MSR codes.

C. Product matrix (PM) framework [11]

In this paper, we utilize the product-matrix framework to obtain the codes with the optimal access bandwidth. Here, we briefly summarize the code constructions obtained by this framework.

1) PM-MBR construction: Let n, k, and d be the given system parameters and $\beta=1$. This translates to $\alpha=d\beta=d$ (cf. (2)). We arrange the $\mathcal{M}=(2kd-k^2+k)/2$ symbols of the underlying file $\mathbf{f}$ in a $d\times d$ symmetric matrix of the following form.

$$M=\begin{bmatrix} \underset{k\times k}{W} & \underset{k\times(d-k)}{T} \\ \underset{(d-k)\times k}{T^t} & \underset{(d-k)\times(d-k)}{0} \end{bmatrix}, \tag{4}$$

where matrix W is a symmetric matrix. Here, for a matrix A, $\underset{a\times b}{A}$ denotes that A is an $a\times b$ matrix. The matrix M is encoded using a $n\times d$ encoding matrix, $\Psi=[\underset{n\times k}{\Phi}\ \ \underset{n\times(d-k)}{\Delta}]$, to obtain the $n\times\alpha$ code matrix $C=\Psi M$. The α symbols in the i-th row are stored in the i-th node of the storage system. In order to ensure the repair and reconstruction properties of the underlying coding scheme, the encoding matrix Ψ is designed to satisfy the following two conditions: 1) Any d rows of Ψ are independent, and 2) Any k rows of Φ are independent. Note that these two requirement can be satisfied by taking Ψ to be a Vandermonde matrix.

Reconstruction of the file: Note that the client contacts k nodes for the reconstruction. Let Ψ_{DC},Φ_{DC} and Δ_{DC} be the sub-matrices related to the k nodes contacted by the client. Thus, the client has access to the $k\times\alpha$ matrix

$$\begin{aligned} X_{DC}=\Psi_{DC}M&=[\underset{k\times k}{\Phi_{DC}}\ \ \underset{k\times(d-k)}{\Delta_{DC}}]M \\ &=[\Phi_{DC}W+\Delta_{DC}T^t\quad \Phi_{DC}T] \end{aligned} \tag{5}$$

Since Φ_{DC} is a full rank square matrix, its inverse can be utilized to get

$$\Phi_{DC}^{-1}X_{DC}=[W+\Phi_{DC}^{-1}\Delta_{DC}T^t\quad T]. \tag{6}$$

This enables us to recover both T and W, which subsequently allows us recover all $\mathcal{M}$ symbols of the stored file (cf. (4)).

Repair of a single node: Assume that the i-th node is being repaired. Note that the following α symbols are lost due to the node failure, $N_i=\psi_i^tM$, where ${\psi_i}^t$ is the i^{th} row of the encoding matrix Ψ. Let $\mathcal{H}=\{j_1,\ldots,j_d\}$ denote the indices of the d helper nodes. Each of the d helper nodes sends the inner product of their content with the vector ψ_i to the newcomer node. As a result, the newcomer node has access to the following information.

$$D_i=\begin{bmatrix}{\psi_{j_1}}^tM\psi_i\\ \vdots\\ {\psi_{j_d}}^tM\psi_i\end{bmatrix}=\begin{bmatrix}{\psi_{j_1}}^t\\ \vdots\\ {\psi_{j_d}}^t\end{bmatrix}M\psi_i=\Psi_{\text{repair}}\ M\psi_i. \tag{7}$$

It follows from the first requirement satisfied by the encoding matrix Ψ that its $d\times\alpha$ sub-matrix Ψ_{repair} is invertible. Thus, by multiplying D_i with $\Psi_{\text{repair}}^{-1}$, we get $\Psi_{\text{repair}}^{-1}D_i=M\psi_i$. This way, the newcomer recovers N_i.

2) PM-MSR construction : The MSR code construction in [11] requires that $2k-2\le d\le n-1$. The $\mathcal{M}$ symbols of the file $\mathbf{f}$ are arranged in a $d\times\alpha$ matrix M. The matrix M takes the following form in the case of MSR code construction.

$$M=\begin{bmatrix}W_1\\ W_2\end{bmatrix}, \tag{8}$$

where W_1 and W_2 are two $(\alpha\times\alpha)$ symmetric matrices. The content of n storage nodes is represented by the $n\times\alpha$ code matrix $C=\Psi M$. Here, Ψ is the $n\times d$ encoding matrix, which again can be taken to be a Vandermonde matrix. We refer the reader to [11] for the descriptions of the reconstruction and the repair operations of this MSR code construction.

III. Min-cut Analysis for Hybrid Storage

In this section, we utilize the modified information flow graph (cf. Section II-B) to understand the trade-off among various parameters of a hybrid cloud storage system. This involves analyzing the min-cut in the flow graph. We note that our min-cut analysis slightly differs from that in [5] due to the presence of local storage and finite capacity edges between the data collector and storage nodes (cf. Remark 1). For a given cut separating the data collector node from the source node, we have three choices to take the contribution of each of the k contacted nodes cut-value into account: 1) multiple input edges to the node with capacity β each, 2) the intermediate edge with capacity α that connects the two sub-nodes associated with the storage node, or 3) the edge with capacity α' that connects the storage node to the data collector node. Thus, the contribution of a contacted storage node to a cut can be $(d-j)\beta$, α, or α', where j denotes the number of previous nodes that have the cut through their input or intermediate edges. Based on this analysis, we obtain the following bound on the file size stored on the system.

Proposition 1. *For the hybrid cloud storage system, the stored file size is upper bounded by*

$$\mathcal{M}\le L+\min_{j=0,\ldots,k}\Big\{(k-j)\alpha'+\sum_{i=0}^{j-1}\min\big\{(d-i)\beta,\alpha\big\}\Big\}. \tag{9}$$

Remark 2. *Since we require the data collector to be able to reconstruct the entire file by combining $k\alpha'$ symbols*

downloaded from the contacted storage nodes with the L symbols stored at the local storage, we have that

$$\mathcal{M} \leq k\alpha' + L. \tag{10}$$

This translates to the following inequalities.

$$\frac{\mathcal{M} - L}{k} \leq \alpha' \leq \alpha. \tag{11}$$

Note that these inequalities can also be obtained from (9) *by taking* $j = 0$.

In order to better understand the bound in Proposition 1, let's consider a function $h : \{0, \ldots, k\} \to \mathbb{R}$ such that

$$h(j) = (k-j)\alpha' + \sum_{i=0}^{j-1} \min((d-i)\beta, \alpha). \tag{12}$$

Note that for any $0 \leq j \leq k-1$, we have

$$h(j+1) - h(j) = -\alpha' + \alpha \geq 0 \tag{13}$$

or

$$h(j+1) - h(j) = -\alpha' + (d-j)\beta. \tag{14}$$

Therefore, the function $h(\cdot)$ is either non-decreasing on the entire domain (if (13) holds on the entire domain) or it is non-decreasing up to a point and then it becomes monotonically decreasing. In both case, the function attains the minimum value at the extremes, which gives us that

$$\min_{j=0,\ldots,k} h(j) = \min\big\{k\alpha', \sum_{i=0}^{k-1} \min\big\{(d-i)\beta, \alpha\big\}\big\}. \tag{15}$$

By combining (15) with the bound in (9), we obtain that

$$\mathcal{M} \leq L + \min\big\{k\alpha', \sum_{i=0}^{k-1} \min\{(d-i)\beta, \alpha\}\big\}. \tag{16}$$

Next, similar to [5], we study the extreme points on the trade-off highlighted by the bound in (16).

A. Minimum storage point

It follows from (11) that we attain the minimum value for the storage when we have

$$\alpha = \alpha' = \frac{\mathcal{M} - L}{k}. \tag{17}$$

In this case, it follows from (16) that

$$\beta = \frac{\mathcal{M} - L}{k(d-k+1)}. \tag{18}$$

Therefore, the following corollary follows from this analysis.

Corollary 1. *The minimum storage point in the hybrid storage model is characterized by the tuple*

$$(\alpha, \beta) = \Big(\frac{\mathcal{M} - L}{k}, \frac{\mathcal{M} - L}{k(d-k+1)}\Big). \tag{19}$$

Moreover, the access bandwidth at the minimum storage is

$$\eta = k\alpha' = \mathcal{M} - L. \tag{20}$$

B. Minimum repair-bandwidth point

In order to minimize the repair-bandwidth, we require that $\alpha \geq d\beta$. By taking $\alpha = d\beta$, it follows from (16) that

$$\mathcal{M} = L + \min\big\{k\alpha', k\alpha - \binom{k}{2}\beta\big\}. \tag{21}$$

Again, in order to minimize the repair-bandwidth we need to assume that $\alpha' \geq \alpha - \frac{k-1}{2}\beta$, which gives us that

$$\mathcal{M} = L + k\alpha - \binom{k}{2}\beta = L + kd\beta - \binom{k}{2}\beta. \tag{22}$$

This gives us the following result.

Corollary 2. *The minimum repair-bandwidth point in the hybrid storage model is characterized by the tuple*

$$(\alpha, \beta) = \Big(\frac{2(\mathcal{M} - L)d}{2kd - k^2 + k}, \frac{2(\mathcal{M} - L)}{2kd - k^2 + k}\Big). \tag{23}$$

Moreover, the access bandwidth at this point satisfies

$$\eta = k\alpha' \geq \mathcal{M} - L. \tag{24}$$

Remark 3. *It follows from Corollary 1 and 2 that the extreme points on the storage vs. repair-bandwidth trade-off for hybrid storage correspond to the MSR and MBR point from [5] with file size $\mathcal{M} - L$, respectively. This also suggests a natural separation based achievability scheme for these points, which stores L symbols of the file in local storage and encodes the $k\alpha' = \mathcal{M} - L$ symbols using an MSR (or MBR) code.*

C. Multiple clients

So far we have considered the scenario where the hybrid cloud supports the clients that have accesses to the same local storage space, which can be modeled by a single client with local storage space of L symbols. Next, we consider the setting with multiple clients that have access to varying amount of local storage space. Assuming that $\{L_1, L_2, \ldots, L_m\}$ denotes the set of different local storage spaces available at these clients, it's straight forward to extend the min-cut analysis over an information flow graph to show that we have

$$\mathcal{M} \leq \min_{l \in [m]} L_l + \min\big\{k\alpha', \sum_{i=0}^{k-1} \min\{(d-i)\beta, \alpha\}\big\}. \tag{25}$$

Accordingly, the minimum storage and the minimum repair-bandwidth points in the presence of multiple clients reduce the tuples

$$(\alpha, \beta) = \Big(\frac{\mathcal{M} - L_{\min}}{k}, \frac{\mathcal{M} - L_{\min}}{k(d-k+1)}\Big), \tag{26}$$

and

$$(\alpha, \beta) = \Big(\frac{2(\mathcal{M} - L_{\min})d}{2kd - k^2 + k}, \frac{2(\mathcal{M} - L_{\min})}{2kd - k^2 + k}\Big), \tag{27}$$

respectively. Here, we use $L_{\min}$ to denote $\min_{l \in [m]} L_l$.

Thus, as expected, the per-node storage and the repair-bandwidth are dictated by the client with the smallest local storage space. However, it is possible to design coding schemes that allow each client to minimize its access

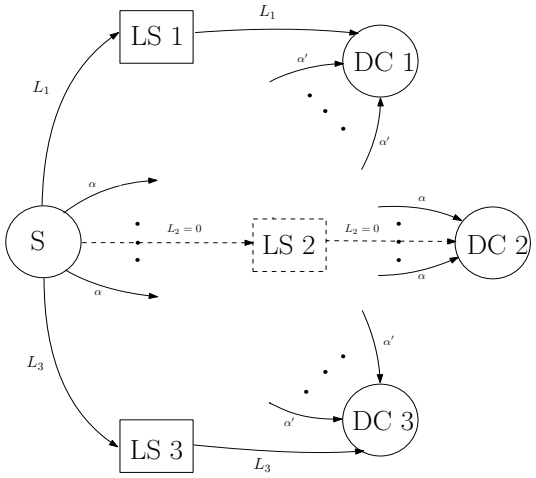

Fig. 3: Min-cut graph for multiple DCs.

bandwidth according to its local storage space. Here, we would also like to point out that enabling different access bandwidths for different clients is not possible with the natural separation scheme (cf. Remark 3).

IV. Construction for MBR case

We now present coding schemes to achieve the minimum bandwidth point on the per node storage vs. repair-bandwidth trade-off. Without loss of generality, we assume that $L_{\min} = \min_{l\in[m]} L_l = 0$, i.e., there exists a client with no local storage space[1]. As discussed in Section III-C, the minimum bandwidth point in this case correspond to the MBR point [5]. In this section, we employ the product-matrix construction from [11] to encode the file to be stored on the global space.

The said repair-bandwidth (cf. (27)) is guaranteed by the repair property of the code. Here, we focus on showing that this code does allow a client to attain the optimal access bandwidth, i.e., $\eta = k\alpha' = \mathcal{M} - L$, where L denotes the size of the local storage space of the client. First, we demonstrate this property for the clients corresponding to $L_{\min} = 0$. We then demonstrate the access bandwidth efficient reconstruction for clients with non-zero local storage space.

A. Without local storage

Assume that the system employs a product-matrix based MBR code to store a file of size $\mathcal{M}$ (cf. Section II-C1). Here, we demonstrate the reconstruction of the entire file by downloading $\alpha' = \frac{\mathcal{M}}{k}$ symbols from each contacted node with the help of an example[2]. We then present the reconstruction process for general parameters.

[1]If $L_{\min} > 0$, then we can separately store $L_{\min}$ symbols from the file at each of the client and encode the remaining $\mathcal{M} - L_{\min}$ symbols using the coding scheme with the reduced local storage space at each client.

[2]In the literature, an MBR code based distributed storage system is allowed to download $k\alpha > \mathcal{M}$ symbols for the reconstruction [5].

Example 1. Let $(n = 6, k = 3, d = \alpha = 4, \beta = 1, \mathcal{M} = 9)$ be the parameters of the underlying MBR codes, under $\mathbb{F}_7$. In this case, the message matrix M takes the following form.

$$M = \begin{bmatrix} u_1 & u_2 & u_3 & u_4 \\ u_2 & u_5 & u_6 & u_7 \\ u_3 & u_6 & u_8 & u_9 \\ u_4 & u_7 & u_9 & 0 \end{bmatrix}. \tag{28}$$

For these parameters, the optimum α' satisfies

$$\alpha' = \frac{\mathcal{M}}{k} = \alpha - \frac{k-1}{2}\beta = 4 - 1 = 3. \tag{29}$$

Thus, we hope to download $\alpha' = 3$ symbols from each of the contacted $k = 3$ nodes during the reconstruction of the file. We assume that we contact the first three nodes during the reconstruction. The last two symbols stored in each of the three contacted nodes give us

$$X_{DC}^{(3,4)} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 1 \\ 1 & 3 & 2 & 6 \end{bmatrix} \begin{bmatrix} u_3 & u_4 \\ u_6 & u_7 \\ u_8 & u_9 \\ u_9 & 0 \end{bmatrix}. \tag{30}$$

Using the ideas utilized in (5) and (6), these symbols are sufficient to recover all the symbols appearing in the last two columns of matrix M. Now, given the access to the second symbol in the first two contacted nodes and the first symbol in the third contacted node, we obtain

$$X_{DC}^{(2)} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 1 \end{bmatrix} \begin{bmatrix} u_2 \\ u_5 \\ u_6 \\ u_7 \end{bmatrix}, \quad X_{DC}^{(1)} = \begin{bmatrix} 1 & 3 & 2 & 6 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix}. \tag{31}$$

Combining the already known u_6 and u_7 with $X_{DC}^{(2)}$ gives us

$$X'^{(2)}_{DC} = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} u_2 \\ u_5 \end{bmatrix}. \tag{32}$$

This enables us to recover u_2 and u_5. Using the same idea, we can get u_1 from $X_{DC}^{(1)}$. Thus, we recover the whole file by downloading only 3 symbols from each contacted node.

Next, we show how to generalize this example for a general product-matrix based MBR code with parameters $(n, k, d, \alpha, \beta = 1)$ and $\alpha' = \frac{\mathcal{M}}{k} = \alpha - \frac{k-1}{2}$. We first focus on the case where k is odd. We then comment on how to handle the settings with even k.

1) Efficient access when k is odd: Again, we assume that the client contacts the first k nodes in the system. By downloading the last $(d-k)$ symbols stored in the k contacted nodes, we get $X_{DC}^{(k+1:d)} = \Phi_{DC}T$. Since Φ_{DC} is a full rank matrix, these symbols allow us to recover T. Similar to Example 1, we can recover the remaining symbols using *backward induction*. Towards this, let $\mathbf{m}_i$ denote the i-th column of M. For $1 \le i \le k$, assume that we know all the symbols included in the columns $\{\mathbf{m}_{i+1}, \ldots, \mathbf{m}_d\}$. Then, by downloading the i-th symbols from any i out of k contacted nodes, we get

$$X_{DC}^{(i)} = \underset{i\times k}{[\Phi_{DC}} \quad \underset{i\times(d-k)}{\Delta_{DC}}\,] \underset{d\times 1}{\mathbf{m}_i}. \tag{33}$$

The previous knowledge of all the symbols in $\{\mathbf{m}_{i+1}, \ldots, \mathbf{m}_d\}$ and the symmetry of M implies that we already know the $d-i$ symbols of $\mathbf{m}_i$. This enables us to get ${X'}_{DC}^{(i)}$ as follows.

$$ {X'}_{DC}^{(i)} = \underset{i\times k}{[\Phi_{DC}} \quad \underset{i\times(d-k)}{\Delta_{DC}}\,] \begin{bmatrix} \underset{i\times 1}{\mathbf{m}_i} \\ 0 \\ \underset{(d-i)\times 1}{} \end{bmatrix} = \underset{i\times i}{\Phi_{DC}} \underset{i\times 1}{\mathbf{m}_i}, \quad (34) $$

where, by abusing the notation, we use $\underset{i\times 1}{\mathbf{m}_i}$ to represent the first i elements of the vector $\mathbf{m}_i$. Now, we can recover $\mathbf{m}_i$ after multiplying ${X'}_{DC}^{(i)}$ by Φ_{DC}^{-1}. Using this backward induction approach, we are able to reconstruct the whole file by downloading

$$ \eta = \sum_{i=1}^{k} i + k(d-k) = k\alpha - \binom{k}{2} \quad (35) $$

symbols, where we need to download $\alpha' = \alpha - \frac{k-1}{2}$ symbols from each of the contacted nodes.

2) Efficient access when k is even: The same ideas explored in Section IV-A1 can be employed here as well. However, the only caveat is that $\alpha' = \alpha - \frac{k-1}{2}$ is not an integer. This issue can be addressed in two potential manners.

(i) **Asymmetric download:** In this solution, we allow the data collector to download $\lceil \alpha' \rceil$ symbols from some $k/2$ contacted nodes, and $\lfloor \alpha' \rfloor$ symbols from the remaining $k/2$ contacted nodes.

(ii) **Sub-packetization:** In this solution, we can divide each symbol into two sub-symbols. We then encode the all $\mathcal{M}$ first sub-symbols and all $\mathcal{M}$ second sub-symbols by treating them as separate files. Now, by carefully utilizing the asymmetric download approach (across two independent encoded files) as discussed above, we can ensure that we download the same amount of data from each of the k contacted nodes.

B. With local storage

We now demonstrate how a client with non-zero local storage can lower the access bandwidth to $\mathcal{M} - L$. In particular, we first revisit Example 1 in the presence of non-zero L.

Example 2. Recall that the parameters of the underlying MBR code are $(n = 6, k = 3, d = \alpha = 4, \beta = 1)$. We assume that the client has local storage of size $L = 3$, where it stores the symbols u_4, u_7, and u_9. For these parameters, the optimum α' satisfies.

$$ \alpha' = \frac{\mathcal{M} - L}{k} = \alpha - \frac{k-1}{2}\beta - \frac{L}{k} = 4 - 1 - 1 = 2. \quad (36) $$

Thus, during an access bandwidth optimal file reconstruction, the client should download only 2 symbols from each of the contacted $k = 3$ nodes. Note that the file reconstruction here can be achieved in the same way as in Example 1 except that here the client does not need to download the last symbol from the contacted nodes. This follows as the client already stores all the symbols appearing in the last column of M.

The file reconstruction with optimal access bandwidth, as highlighted in Example 2, can be generalized for other system parameters in a straightforward manner. Here, we briefly comment on this. Given the local storage size L and k is odd, there are two possible cases:

1) k **divides** L**:** In this case, we can store any L symbols. Now, using the same ideas as in (32), the previous knowledge of the stored symbols help us obtain saving of L symbols during download.
2) k **does not divide** L**:** We can write L as $L = sk + r$, where $s = \lfloor L/k \rfloor$ and $0 \leq r < k$. In this case , the $\alpha' = \frac{\mathcal{M}-L}{k} = \alpha - \frac{k-1}{2}\beta - \frac{L}{k}$ is non-integer. Similar to Section IV-A2, we can overcome this problem in two possible ways.
   (i) **Asymmetric download:** During the file reconstruction, we download $\lceil \alpha' \rceil$ symbols from r contacted nodes, and $\lfloor \alpha' \rfloor$ symbols from the remaining $k - r$ contacted nodes.
   (ii) **Sub-packetization:** We divide each symbol into k sub-symbols from a sub-field. Then, we separately encode all sub-symbols with the same order as one file. This way we get k independent codewords of the underlying MBR code. Now, by carefully utilizing the asymmetric download approach (across k independent encoded files) as discussed above we can ensure that we download the same amount of data from each of the k contacted nodes.

V. Construction for MSR case

In this section, we briefly comment on the issue of access bandwidth efficient file reconstruction at the minimum storage point. Again, without loss of generality we assume a setup with multiple clients such that $L_{\min} = 0$. For the MSR case, the access bandwidth for a client without any local storage space is trivially $\eta = k\alpha = \mathcal{M}$. Therefore, we only focus on the file reconstruction at the clients with non-zero local storage space. In this paper, we restrict ourselves to product-matrix based MSR codes. However, similar results can be easily obtained for other MSR codes as well.

Let's consider the product-matrix based MSR codes with generic parameters $(n = 2k-1, k, d = 2k-2, \alpha = k-1, \beta = 1)$. In order to demonstrate access bandwidth optimal file reconstruction, we first focus on the case with $L = ik$ for $i \geq 1$. The setting where k does not divide L can be handle as illustrated in Section IV-B.

Assuming that $L = ik$, for $i \in [\alpha] = [k-1]$, we can achieve the optimal access bandwidth by downloading $\alpha' = \alpha - i$ symbols from k nodes during the file reconstruction. We assume that the client downloads the first $\alpha' = \alpha - i$ symbols stored in the k contacted nodes. Therefore, the client stores the $2\binom{i+1}{2}$ symbols located in the two $i \times i$ symmetric lower right corners of the sub-matrices of W_1 and W_2 (cf. (8)) as these symbols do not appear in the downloaded symbols from the contacted nodes. Moreover, the client utilizes the remaining local storage space of size $L - 2\binom{i+1}{2} = ik - 2\binom{i+1}{2} = (k-2) + (k-4) + \cdots + (k-2i)$ to store $k - 2j$

symbols from the j-th column of M, $\forall 1 \leq j \leq i$. Using the following equation,

$$X_{DC}^{(1)} = \underset{k\times d}{\Psi_{DC}} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ u_{k-1} \\ \vdots \\ u_{2k-2} \end{bmatrix} = \underset{k\times k}{\Psi'_{DC}} \begin{bmatrix} u_{k-1} \\ \vdots \\ u_{2k-2} \end{bmatrix}, \qquad (37)$$

and utilizing the fact that $\underset{k\times d}{\Psi'_{DC}}$ is full rank, the client can recover the unknown k symbols located in the first column of M. By symmetry of W_1 and W_2, this enables the client to know 2 symbols in each of other columns. Thus, for the second column, the client now knows $k-2$ symbols and recover the unknown k symbols with the help of the associated linear combinations downloaded from the k contacted nodes. Continuing this idea till the i-th column of M, the client has access to ki symbols from the local storage and already recovered ki symbols from the first i columns of M. The remaining $\alpha' - i = k - 1 - 2i$ symbols per contacted node provide $k(k-1-2i)$ independent linear combinations, which are sufficient to recover the required $k\alpha - 2ki = k(k-1-2i)$ remaining required symbols. Hence, the client is able to reconstruct the whole file with the optimal access bandwidth $\eta = k\alpha' = k\alpha - ik = \mathcal{M} - L$.

VI. Conclusion

We carried out a systematic study of trade-off among various parameters of a hybrid cloud storage system. In particular, we focused on the utilization of the local storage space to reduce access-bandwidth, the amount of information that needs to be downloaded from the global (public) storage space during a file reconstruction, while ensuring that the global storage space enables bandwidth efficient repair of failed storage nodes. This corresponds to a joint optimization of local and global resources. In this paper, we particularly focused on the two extreme points of the storage vs. repair-bandwidth trade-off curve and utilized the product-matrix based codes to attain these points. It's an interesting problem to explore access-bandwidth efficient coding schemes for intermediate points on the trade-off. Similarly, the hybrid storage model considered here (especially the information flow graph) can be further modified to incorporate various system constraints, including the efficient repair of local storage space.

References

[1] "Massive Amazon cloud service outage disrupts sites," https://www.usatoday.com/story/tech/news/2017/02/28/amazons-cloud-service-goes-down-sites-scramble/98530914/, [Online; accessed 10 September 2017].

[2] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds," in *Proc. of CCS*, Nov. 2009, pp. 199–212.

[3] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Cross-vm side channels and their use to extract private keys," in *Proc. of CCS*, Oct. 2012, pp. 305–316.

[4] O. Khan, R. Burns, J. Park, and C. Huang, "In search of i/o-optimal recovery from disk failures," in *Proc. of HotStorage*, June 2011, pp. 6–6.

[5] A. G. Dimakis, P. Godfrey, Y. Wu, M. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *IEEE Trans. Inf. Theory*, vol. 56, no. 9, pp. 4539–4551, Sept. 2010.

[6] P. Gopalan, C. Huang, H. Simitci, and S. Yekhanin, "On the locality of codeword symbols," *IEEE Trans. Inf. Theory*, vol. 58, no. 11, pp. 6925–6934, Nov 2012.

[7] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin, in *Proc. of ATC*, June 2012, pp. 15–26.

[8] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur, "Xoring elephants: novel erasure codes for big data," in *Proc. of VLDB*, Aug. 2013, pp. 325–336.

[9] S. Muralidhar, W. Lloyd, S. Roy, C. Hill, E. Lin, W. Liu, S. Pan, S. Shankar, V. Sivakumar, L. Tang, and S. Kumar, "f4: Facebook's warm BLOB storage system," in *Proc. of OSDI*, Oct. 2014, pp. 383–398.

[10] K. Rashmi, N. B. Shah, D. Gu, H. Kuang, D. Borthakur, and K. Ramchandran, "A Hitchhikers guide to fast and efficient data reconstruction in erasure-coded data centers," in *Proc. of SIGCOMM*, Aug. 2014, pp. 331–342.

[11] K. V. Rashmi, N. B. Shah, and P. V. Kumar, "Optimal exact-regenerating codes for distributed storage at the MSR and MBR points via a product-matrix construction," *IEEE Trans. Inf. Theory*, vol. 57, no. 8, pp. 5227–5239, Aug. 2011.

[12] D. S. Papailiopoulos and A. G. Dimakis, "Locally repairable codes," *IEEE Trans. Inf. Theory*, vol. 60, no. 10, pp. 5843–5855, Oct 2014.

[13] A. S. Rawat, O. O. Koyluoglu, N. Silberstein, and S. Vishwanath, "Optimal locally repairable and secure codes for distributed storage systems," *IEEE Trans. Inf. Theory*, vol. 60, no. 1, pp. 212–236, Jan 2014.

[14] I. Tamo and A. Barg, "A family of optimal locally recoverable codes," *IEEE Trans. Inf. Theory*, vol. 60, no. 8, pp. 4661–4676, Aug 2014.

[15] M. Ye and A. Barg, "Explicit constructions of high-rate MDS array codes with optimal repair bandwidth," *IEEE Trans. Inf. Theory*, vol. 63, no. 4, pp. 2001–2014, Apr. 2017.

[16] ——, "Explicit constructions of optimal-access MDS codes with nearly optimal sub-packetization," *CoRR*, vol. abs/1605.08630, 2016. [Online]. Available: http://arxiv.org/abs/1605.08630

[17] I. Tamo, Z. Wang, and J. Bruck, "Zigzag codes: MDS array codes with optimal rebuilding," *IEEE Trans. Inf. Theory*, vol. 59, no. 3, pp. 1597–1616, Mar. 2013.

[18] B. Sasidharan, M. Vajha, and P. V. Kumar, "An explicit, coupled-layer construction of a high-rate MSR code with low sub-packetization level, small field size and all-node repair," *CoRR*, vol. abs/1607.07335, 2016. [Online]. Available: http://arxiv.org/abs/1607.07335

[19] "Cloud-enabled data center for dummies," http://download.1105media.com/pub/101communications/files/averesystems_cloudenableddatacenterfordummies.pdf, [Online; accessed 24 September 2017].

[20] M. A. Maddah-Ali and U. Niesen, "Fundamental limits of caching," *IEEE Trans. Inf. Theory*, vol. 60, no. 5, pp. 2856–2867, May 2014.

[21] ——, "Coding for caching: fundamental limits and practical challenges," *IEEE Commun. Mag.*, vol. 54, no. 8, pp. 23–29, August 2016.

[22] T. Luo, V. Aggarwal, and B. Peleato, "Coded caching with distributed storage," *CoRR*, vol. abs/1611.06591, 2016. [Online]. Available: http://arxiv.org/abs/1611.06591

[23] V. Aggarwal, Y. F. R. Chen, T. Lan, and Y. Xiang, "Sprout: A functional caching approach to minimize service latency in erasure-coded storage," in *Proc. of IEEE ICDCS*, June 2016, pp. 753–754.

[24] I. Samy, G. Calis, and O. O. Koyluoglu, "Secure regenerating codes for hybrid cloud storage systems," in *Proc. of IEEE ISIT*, June 2017, pp. 2208–2212.