Capacity of Clustered Distributed Storage

Jy-yong Sohn, Beongjun Choi, Sung Whan Yoon, and Jaekyun Moon
School of Electrical Engineering
Korea Advanced Institute of Science and Technology
Daejeon, 34141, Republic of Korea
Email: {jysohn1108, bbzang10, shyoon8}@kaist.ac.kr, jmoon@kaist.edu

***Abstract*—A new system model reflecting the clustered structure of distributed storage is suggested to investigate bandwidth requirements for repairing failed storage nodes. Large data centers with multiple racks/disks or local networks of storage devices (e.g. sensor network) are good applications of the suggested cluster-based model. In realistic scenarios involving clustered storage structures, repairing storage nodes using intact nodes residing in other clusters is more bandwidth-consuming than restoring nodes based on information from intra-cluster nodes. Therefore, it is important to differentiate between intra-cluster repair bandwidth and cross-cluster repair bandwidth in modeling distributed storage. Capacity of the suggested model is obtained as a function of fundamental resources of distributed storage systems, namely, storage capacity, intra-cluster repair bandwidth and cross-cluster repair bandwidth. Based on the capacity expression, feasible sets of required resources which enable reliable storage are analyzed. It is shown that the cross-cluster traffic can be minimized to zero (i.e., local repair within a cluster becomes possible) by allowing extra resources on storage capacity and intra-cluster repair bandwidth, according to a law specified in a closed-form. Moreover, trade-off between cross-cluster traffic and intra-cluster traffic is observed for sufficiently large storage capacity.**

I. Introduction

Many enterprises use cloud storage systems in order to support massive amounts of data storage requests from clients. In the emerging Internet-of-Thing (IoT) era, the number of devices which generate data and connect to the network increases exponentially, so that efficient management of data center becomes a formidable challenge. Moreover, since a cloud storage system consists of inexpensive commodity disks, failure events occur frequently, degrading system reliability [1].

In order to ensure reliability of cloud storage, distributed storage systems (DSSs) with erasure coding have been considered to improve tolerance against storage node failures [2]–[4]. In such systems, the original file is encoded and distributed into multiple storage nodes. When a node fails, a newcomer node regenerates the failed node by contacting a number of survived nodes. This causes traffic burden across the network taking up significant repair bandwidth. The pioneering work of [5] on distributed storage found the system capacity $\mathcal{C}$, the maximum amount of reliably storable data given the storage size of each node as well as repair bandwidth. A fundamental trade-off between storage node size and repair bandwidth was obtained, which ensures a reliable storage system with the maximum-distance-separable (MDS) property (i.e., any k out of n storage nodes can be accessed to recover the original file). The authors related the failure-repair process of a DSS with the multi-casting problem in network information theory, and exploited the fact that cut-set bound is achievable by network coding [6].

This fundamental result is based on the assumption of a homogeneous system, i.e., each node has the same storage size and repair bandwidth. However, storage nodes are actually dispersed into multiple clusters (called disks or racks) in data centers [7]–[9], allowing high reliability against both node and rack failures. In this clustered system, repairing a failed node gives rise to both intra-cluster and cross-cluster repair traffic. The cross-rack communication bandwidth is typically oversubscribed by a factor of 5 to 20 in practical systems [10], *i.e.*, the cross-rack traffic demand is 5 to 20 times greater than the available bandwidth. Thus, reducing the cross-rack repair traffic is one of the major implementation issues; a new system model which reflects the imbalance between intra-cluster and cross-cluster repair bandwidth is required.

Main Contributions: This paper suggests a new system model for *clustered DSS* to reflect the clustered nature of real distributed storage systems wherein an imbalance exists between intra and cross-cluster repair burdens. This model can be applied to not only large data centers, but also local networks of storage devices such as the sensor networks or home clouds which are expected to be prevalent in the IoT era. This model is also more general in the sense that when the intra- and cross-cluster repair bandwidths are set equal, the resulting structure reduces to the original DSS model of [5].

Storage capacity of the clustered DSS is obtained as a function of node storage size, intra-cluster repair bandwidth and cross-cluster repair bandwidth. The existence of the cluster structure manifested as the imbalance between intra/cross-cluster traffics makes the capacity analysis challenging; The main analysis given in [5] cannot be directly extended to handle the problem at hand. We show that symmetric repair (obtaining the same amount of information from each helper node) is optimal in the sense of maximizing capacity given the storage node size and total repair bandwidth, as also shown in

This work is in part supported by the National Research Foundation of Korea under Grant No. 2016R1A2B4011298, and in part supported by the ICT R&D program of MSIP/IITP [2016-0-00563, Research on Adaptive Machine Learning Technology Development for Intelligent Autonomous Digital Companion].

[11] for the case of varying repair bandwidth across the nodes. However, we stress that in most practical scenarios, the need is greater for reducing cross-cluster communication burden, and we show that this is possible by trading with reduced overall storage capacity and/or increasing intra-repair bandwidth.

The behavior of the clustered DSS is also observed based on the derived capacity expression. First, the condition for *zero* cross-cluster repair bandwidth is obtained, which enables local repair via intra-cluster communication only. In order to support local repair while preserving the MDS property, extra amounts of resources are required, according to a precise mathematical rule derived in a closed-form. Secondly, trade-off between cross-cluster repair bandwidth and intra-cluster repair bandwidth is discussed. The repair bandwidth resource is required to ensure reliability of a file against successive failure events; the amounts of intra-/cross-cluster repair bandwidth can be chosen in the trade-off curve, depending on the system constraint.

Related works: In order to reflect the non-homogeneous structure of storage nodes, several researchers in the literature aimed to analyze practical distributed storage systems. A heterogeneous model was considered in [11], where storage size of each node and repair bandwidth for each newcomer node is generally non-uniform. Upper/lower capacity bounds for the heterogeneous DSS are obtained in [11]. A asymmetric repair process is considered in [12], coining *cheap* versus *expensive bandwidths*, depending on the amount of data that can be transfered to any newcomer. This is different from our analysis where we adopt a notion of cluster and introduce imbalance between intra- and cross-cluster repair burdens.

In [13], the idea of [12] is developed to a two-rack system, by setting the communication burden within a rack much lower than the burden across different racks, similar to our analysis. However, they classified the number of helper nodes by each rack. In other words, any failed node is repaired by d_1 helper nodes in 1^{st} rack and d_2 helper nodes in 2^{nd} rack, irrespective of the location of the failed node. On the other hand, the present paper classifies the number of helper nodes by their locations relative to the failed node. A failed node is repaired by d_I helper nodes within the same rack and d_c helper nodes in other racks. Compared to [13], the setting in our work allows insights into tradeoffs between the intra- and cross-rack repair bandwidth. Moreover, [13] focused on a repair cost function based on a weighted sum of two types of repair bandwidths, whereas this paper investigates the trade-off between the two different repair bandwidth types as well as the storage node size. Finally, some other works focused on the code design applicable to multi-rack DSSs [14], [15].

Organization: This paper is organized as follows. Section II describes basic preliminary materials about distributed storage systems and the information flow graph, an efficient tool for analyzing DSS. A new system model for the clustered DSS is suggested in Section III, while the capacity of the suggested model is obtained in Section IV. Based on the derived capacity expression, the key nature of the clustered DSS is unveiled in Section V. Finally, Section VI draws the conclusion.

II. Background

A. Distributed Storage System

Distributed storage systems have been considered as a candidate for storing data, which maintains reliability by means of erasure coding [16]. The original data file is spread into n unreliable nodes, each with storage size α. When a node fails, it is regenerated by contacting $d < n$ helper nodes and obtaining a particular amount of data, β, from each helper node. The amount of communication burden allocated for one failure event is called the repair bandwidth, denoted as $\gamma = d\beta$. When the client requests a retrieval of the original file, assuming all failed nodes have been repaired, access to any k out of n nodes must guarantee a file recovery. The ability to recover the original data using any $k < n$ out of n nodes is based on the MDS property. Distributed storage systems can be used in many applications such as large data centers, peer-to-peer storage systems and wireless sensor networks [5].

B. Information Flow Graph

Information flow graph is a useful tool to analyze the amount of information flow from source to data collector in a DSS [5]. It is a directed graph consisting of three types of nodes: data source S, data collector DC, and storage nodes x^i_{in}, x^i_{out} as shown in Fig. 1. Storage node x^i can be viewed as consisting of 'input-node' x^i_{in} and 'output node' x^i_{out}, which are responsible for the incoming and outgoing edges, respectively. x^i_{in} and x^i_{out} are connected by a directed edge with capacity identical to the storage size α of node x^i.

Data from the source is stored into n nodes. This process is represented by n edges going from S to $\{x^i\}_{i=1}^n$, where each edge capacity is set to infinity. A failure/repair process in a DSS can be described as follows. When a node x^j fails, a new node x^{n+1} joins the graph by connecting edges from d survived nodes, where each edge has capacity β. After all repairs are done, data collector DC chooses arbitrary k nodes to retrieve data, as illustrated by the edges connected from k survived nodes with infinite edge capacity. Fig. 1 gives an example of information flow graph representing a distributed storage system with $n = 4, k = 3, d = 3$.

For a given flow graph G, a cut between S and DC is defined as a subset C of edges which satisfies the following: every directed path from S to DC includes at least one edge in C. The cut between S and DC having the smallest total sum of edge capacities is called minimum cut. A min-cut value is defined as the sum of edge capacities included in the minimum cut.

III. System Model

A. Clustered Distributed Storage System

A distributed storage system with multiple clusters is shown in Fig. 2. Data from the source S is stored at n nodes which are grouped into L clusters. The number of nodes at each cluster is fixed and denoted as $n_I = n/L$. The storage size of each node is denoted as α. When a node fails, a newcomer node is regenerated by contacting d_I helper

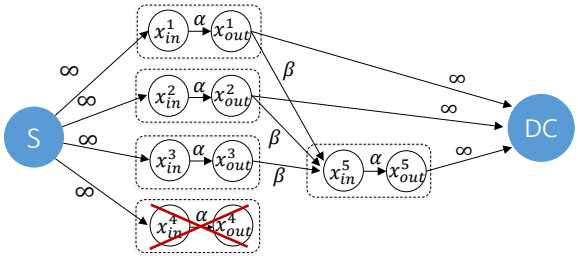

Fig. 1: Information flow graph ($n = 4, k = 3, d = 3$)

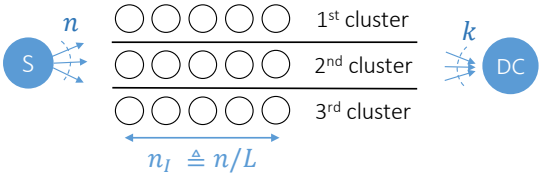

Fig. 2: Clustered distributed storage system ($n = 15, L = 3$)

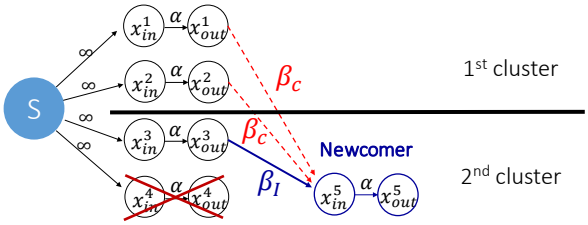

Fig. 3: Repair process in clustered DSS ($n = 4, L = 2, d_I = 1, d_c = 2$)

nodes within the same cluster, and d_c helper nodes from other clusters. The amount of data a newcomer node receives within the same cluster is $\gamma_I = d_I\beta_I$ (each node equally contributes β_I), and that from other clusters is $\gamma_c = d_c\beta_c$ (each node equally contributes β_c). Fig. 3 illustrates an example of information flow graph representing the repair process in a clustered DSS. The suggested heterogeneous model reduces to the homogeneous model in [5], in the case of $\beta_I = \beta_c$. We assume that d_c and d_I have the maximum possible values ($d_c = n - n_I, d_I = n_I - 1$), since this setting most efficiently utilizes the system resources: node storage size and repair bandwidth [5]. A data collector DC contacts any k out of n nodes in the clustered DSS.

B. Problem Formulation

Consider a clustered DSS with fixed n, k, L values. In this model, we want to find the set of *feasible* parameters $(\alpha, \gamma_I, \gamma_c)$ which enables storing data of size $\mathcal{M}$. In order to find the feasible set, min-cut analysis on the information flow graph is required, similar to [5]. Depending on the failure-repair process and k nodes contacted by DC, various information flow graphs can be obtained.

Let $\mathcal{G}$ be the set of all possible flow graphs. Denote the graph with minimum min-cut as G^*. Consider arbitrary information flow graph $G \in \mathcal{G}$. Based on the max-flow min-cut theorem

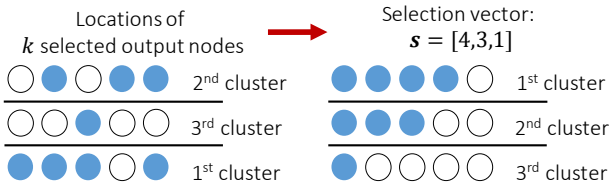

Fig. 4: Obtaining the selection vector for given k output nodes ($n = 15, k = 8, L = 3$)

in [6], the maximum information flow from the source to the data collector for G is greater than equal to

$$\mathcal{C}(\alpha, \gamma_I, \gamma_c) \triangleq \text{min-cut of } G^* \tag{1}$$

which is called the *capacity* of the system. In order to send data $\mathcal{M}$ from the source to the data collector, $\mathcal{C} \geq \mathcal{M}$ should be satisfied. Moreover, if $\mathcal{C} \geq \mathcal{M}$ is satisfied, there exists a linear network coding scheme [6] to store a file with size $\mathcal{M}$. Therefore, the set of $(\alpha, \gamma_I, \gamma_c)$ points which satisfies $\mathcal{C} \geq \mathcal{M}$ is *feasible* in the sense of reliably storing the original file of size $\mathcal{M}$.

This paper first finds the min-cut minimizer $G^* \in \mathcal{G}$ and then obtains capacity $\mathcal{C}(\alpha, \gamma_I, \gamma_c)$,as will be shown in Section IV. Given that the typical intra-cluster communication bandwidth is larger than the cross-cluster bandwidth in real systems, we assume $\beta_I \geq \beta_c$ throughout the present paper.

IV. Capacity of Clustered DSS

In this section, a closed-form solution for capacity $\mathcal{C}(\alpha, \gamma_I, \gamma_c)$ in (1) is obtained by specifying the min-cut minimizer G^*. For a fixed $G \in \mathcal{G}$, let $\{x_{out}^{c_i}\}_{i=1}^{k}$ be the set of output nodes contacted by the data collector, ordered by topological sorting. Note that every directed acyclic graph can be topologically sorted [17], where vertex u is followed by vertex v if there exists a directed edge from u to v. Depending on the selection of k output nodes (among n nodes) and ordering of the selected nodes, different kinds of flow graphs with possibly different min-cut values are obtained. Therefore, in order to obtain min-cut minimizing flow graph G^*, we need to specify 1) the optimal ordering method of the given k output nodes and 2) the optimal selection method of k output nodes, which are stated as Lemmas 1 and 2, respectively. A selection method is mathematically expressed as a selection vector $\boldsymbol{s}$, while an ordering method is expressed as an ordering vector $\boldsymbol{\pi}$, defined in the following subsection.

A. Selection vector, ordering vector and min-cut

Definition 1. *Let arbitrary k nodes be selected as the output nodes $\{x_{out}^{c_i}\}_{i=1}^{k}$. Label each cluster by the number of selected nodes in a descending order. In other words, the 1^{st} cluster contains a maximum number of selected nodes, and the L^{th} cluster contains a minimum number of selected nodes. Under this setting, define the selection vector $\boldsymbol{s} = [s_1, s_2, \cdots, s_L]$ where s_i is the number of selected nodes in the i^{th} cluster.*

For the selected k output nodes, the corresponding selection vector $\boldsymbol{s}$ is illustrated in Fig. 4. From the definition of selection vector, the set of possible selection vectors can be specified as follows.

$$\mathcal{S} \triangleq \{\boldsymbol{s} = [s_1, \cdots, s_L] : 0 \leq s_i \leq n_I, s_{i+1} \leq s_i, \sum_{i=1}^{k} s_i = k\}$$

By using Definition 1, every selection of k nodes can be assigned to a selection vector $\boldsymbol{s} \in \mathcal{S}$. Note that even though $\binom{n}{k}$ different selections exist, the min-cut value is only determined by the corresponding selection vector $\boldsymbol{s}$. To be specific, consider different selections ς_1, ς_2 assigned to the same $\boldsymbol{s}$ vector. Then, every possible flow graph originated from ς_1 is isomorphic to an element of possible flow graphs originated from ς_2, and vice versa. Thus, ς_1 and ς_2 have the identical set of min-cut values. Therefore, comparing the min-cut values of all $|\mathcal{S}|$ possible selection vectors $\boldsymbol{s}$ is enough; it is not necessary to compare the min-cut values of $\binom{n}{k}$ selection methods.

Now, we define the ordering vector $\boldsymbol{\pi}$ for a given selection vector $\boldsymbol{s}$.

Definition 2. *Let the locations of k output nodes $\{x_{out}^{c_i}\}_{i=1}^{k}$ be fixed, with a corresponding selection vector $\boldsymbol{s} = [s_1, \cdots, s_L]$. Then, for arbitrary ordering of k output nodes, define the ordering vector $\boldsymbol{\pi} = [\pi_1, \cdots, \pi_k]$ where π_i is the index of cluster which contains $x_{out}^{c_i}$.*

For a given $\boldsymbol{s}$, the ordering vector $\boldsymbol{\pi}$ corresponding to an arbitrary ordering of k nodes is illustrated in Fig. 5. In this figure (and the following figures in this paper), the number i written inside each node means that the node is x^{c_i}.

From the definition, an ordering vector $\boldsymbol{\pi} \in \Pi(\boldsymbol{s})$ has s_l components with value l, for all $l \in \{1, \cdots, L\}$. The set of possible ordering vectors can be specified as follows.

$$\Pi(\boldsymbol{s}) = \{\boldsymbol{\pi} = [\pi_1, \cdots, \pi_k] : \sum_{i=1}^{k} \mathbb{1}_{\pi_i = l} = s_l \; \forall l \in \{1, \cdots, L\}\}$$

Here, $\mathbb{1}_{\pi_i = l}$ is an indicator function which has value 1 if $\pi_i = l$, and 0 otherwise. Note that for given k selected nodes, there exists $k!$ different ordering methods. However, the min-cut value is only determined by the corresponding ordering vector $\boldsymbol{\pi} \in \Pi(\boldsymbol{s})$ (based on flow graph analysis, similar to compressing $\binom{n}{k}$ selection methods to $|\mathcal{S}|$ selection vectors). Therefore, comparing the min-cut values of all possible ordering vectors $\boldsymbol{\pi}$ is enough; it is not necessary to compare the min-cut values of all $k!$ ordering methods.

Now, we express the min-cut value as a function of selection vector $\boldsymbol{s}$ and ordering vector $\boldsymbol{\pi}$.

Proposition 1. *For a given $\boldsymbol{s}$, consider arbitrary ordering vector $\boldsymbol{\pi} \in \Pi(\boldsymbol{s})$, where $\boldsymbol{\pi} = [\pi_1, \cdots, \pi_k]$. Then, the minimum min-cut value among possible flow graphs is*

$$c_{min}(\boldsymbol{s}, \boldsymbol{\pi}) = \sum_{i=1}^{k} \min\{\omega_i(\boldsymbol{\pi}), \alpha\} \tag{2}$$

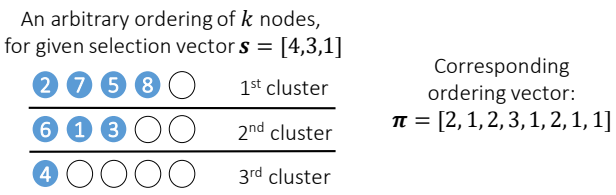

Fig. 5: Obtaining the ordering vector for given an arbitrary order of k output nodes ($n = 15, k = 8, L = 3$)

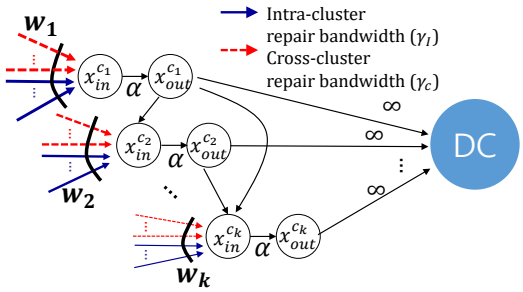

Fig. 6: Information flow graph for obtaining min-cut

where

$$\omega_i(\boldsymbol{\pi}) = a_i(\boldsymbol{\pi})\beta_I + (n - i - a_i(\boldsymbol{\pi}))\beta_c \tag{3}$$

$$a_i(\boldsymbol{\pi}) = n_I - 1 - \sum_{j=1}^{i-1} \mathbb{1}_{\pi_j = \pi_i}. \tag{4}$$

Denote $\omega_i(\boldsymbol{\pi})$ in (3) as the i^{th} *weight value* of the given ordering vector $\boldsymbol{\pi}$. Here, we provide a sketch of the proof for Proposition 1. The formal proof will be given elsewhere. Consider an arbitrary selection vector $\boldsymbol{s}$ and an ordering vector $\boldsymbol{\pi}$. Then, the k nodes $\{x^{c_i}\}_{i=1}^{k}$ connected to the data collector are specified. Consider an information flow graph G illustrated in Fig. 6, which satisfies the following: for every i, j satisfying $1 \leq i < j \leq k$, there is a directed edge from $x_{out}^{c_i}$ to $x_{out}^{c_j}$. In this graph, the sum of capacities of edges coming into $x_{in}^{c_i}$ (except those from $\{x_{out}^{c_t}\}_{t=1}^{i-1}$) turns out to be ω_i expressed in (3).

Consider a set C of edges generated as follows. For all i satisfying $1 \leq i \leq k$, compare values of ω_i and α. If $\omega_i \leq \alpha$, include the edges coming into $x_{in}^{c_i}$ (except those from $\{x_{out}^{c_t}\}_{t=1}^{i-1}$) in the set C. Otherwise, include the edge from $x_{in}^{c_i}$ to $x_{out}^{c_i}$ to the set C. Then, the generated set C becomes a cut separating the source and the data collector. The sum of capacities of edges included in C is obtained as $\sum_{i=1}^{k} \min\{\omega_i, \alpha\}$ in (2). Finally, any information flow graphs for given selection vector $\boldsymbol{s}$ and ordering vector $\boldsymbol{\pi}$ are shown to have min-cut values greater than or equal to RHS of (2), which completes the proof.

We now state the useful property of ω_i defined in (3).

Proposition 2. *Let n, k, L and selection vector $\boldsymbol{s}$ be fixed. Then, $\sum_{i=1}^{k} \omega_i(\boldsymbol{\pi})$ is constant irrespective of the ordering vector $\boldsymbol{\pi} \in \Pi(\boldsymbol{s})$.*

Algorithm 1 Generate vertical ordering $\boldsymbol{\pi}_v$

Input: $\boldsymbol{s} = [s_1, \cdots, s_L]$
Output: $\boldsymbol{\pi}_v = [\pi_1, \cdots, \pi_k]$
Initialization: $l \leftarrow 1$
for $i = 1$ to k **do**
  if $s_l = 0$ **then**
    $l \leftarrow 1$
  else
    $\pi_i \leftarrow l$; $s_i \leftarrow s_i - 1$; $l \leftarrow l + 1$
  end if
end for

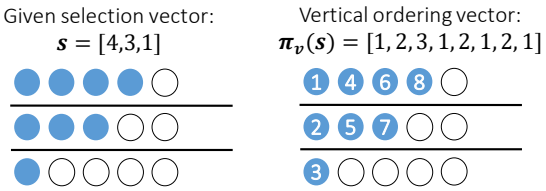

Fig. 7: The vertical ordering vector $\boldsymbol{\pi}_v$ for given selection vector $\boldsymbol{s} = [4, 3, 1]$ (for $n = 15, k = 8, L = 3$ case)

Proof. Consider a fixed selection vector $\boldsymbol{s} = [s_1, \cdots, s_L]$. For an arbitrary ordering vector $\boldsymbol{\pi} \in \Pi(\boldsymbol{s})$, let $b_i(\boldsymbol{\pi}) = n - i - a_i(\boldsymbol{\pi})$ where $a_i(\boldsymbol{\pi})$ is at (4). For simplicity, we denote $a_i(\boldsymbol{\pi})$ and $b_i(\boldsymbol{\pi})$ as a_i and b_i, respectively. Then,

$$c_0 \triangleq \sum_{i=1}^{k}(a_i + b_i) = \sum_{i=1}^{k}(n - i), \tag{5}$$

where c_0 is constant. Note that $\sum_{i=1}^{k} a_i = k(n_I - 1) - \sum_{i=1}^{k}\sum_{j=1}^{i-1} \mathbb{1}_{\pi_j = \pi_i}$. Also, from the definition of $\Pi(\boldsymbol{s})$, the ordering vector $\boldsymbol{\pi}$ has s_l components with value l, for all $l \in \{1, \cdots, L\}$. Let $I_l \triangleq \{i \in \{1, \cdots, k\} : \pi_i = l\}$ for $l = 1, \cdots, L$. Then, $\sum_{i \in I_l}\sum_{j=1}^{i-1} \mathbb{1}_{\pi_j = \pi_i} = 0 + 1 + \cdots + (s_l - 1)$. Therefore,

$$c_1 \triangleq \sum_{i=1}^{k}\sum_{j=1}^{i-1} \mathbb{1}_{\pi_j = \pi_i} = \sum_{l=1}^{L}\sum_{i \in I_l}\sum_{j=1}^{i-1} \mathbb{1}_{\pi_j = \pi_i} = \sum_{l=1}^{L}\sum_{t=0}^{s_l - 1} t,$$

where c_1 is constant. Thus, $\sum_{i=1}^{k} a_i = k(n_I - 1) - c_1$ is also a constant. From (5), we get $\sum_{i=1}^{k} b_i = c_0 - \sum_{i=1}^{k} a_i$, which is another constant. Therefore, $\sum_{i=1}^{k} \omega_i = (\sum_{i=1}^{k} a_i)\beta_I + (\sum_{i=1}^{k} b_i)\beta_c$ is constant for every ordering vector $\boldsymbol{\pi} \in \Pi(\boldsymbol{s})$. □

B. Two Lemmas Supporting the Main Theorem

Now, we state our first main Lemma, which specifies the optimal ordering vector $\boldsymbol{\pi}$ which minimizes $c_{min}(\boldsymbol{s}, \boldsymbol{\pi})$ for an arbitrary selection vector $\boldsymbol{s}$.

Lemma 1. *Let $\boldsymbol{s} \in \mathcal{S}$ be an arbitrary selection vector. Then, the vertical ordering vector $\boldsymbol{\pi}_v$ obtained by Algorithm 1 minimizes the min-cut. In other words, $c_{min}(\boldsymbol{s}, \boldsymbol{\pi}_v) \leq c_{min}(\boldsymbol{s}, \boldsymbol{\pi})$ holds for arbitrary $\boldsymbol{\pi} \in \Pi(\boldsymbol{s})$.*

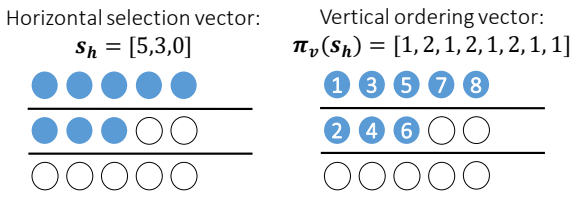

Fig. 8: The optimal selection vector $\boldsymbol{s}_h$ and the optimal ordering vector $\boldsymbol{\pi}_v$ (for $n = 15, k = 8, L = 3$ case)

The vertical ordering vector is illustrated in Fig. 7, for a given selection vector $\boldsymbol{s}$ as an example. For $\boldsymbol{s} = [4, 3, 1]$, Algorithm 1 produces the corresponding vertical ordering vector $\boldsymbol{\pi}_v = [1, 2, 3, 1, 2, 1, 2, 1]$. Note that the order of $k = 8$ output nodes is illustrated in Fig. 7, as the numbers inside each node. Although the vertical ordering vector $\boldsymbol{\pi}_v$ depends on the selection vector $\boldsymbol{s}$, we use simplified notation $\boldsymbol{\pi}_v$.

Due to space limitation, here we just sketch the proof of Lemma 1. The formal proof will be given elsewhere. Let a selection vector $\boldsymbol{s}$ be given. Consider a running sum $S_t(\boldsymbol{\pi}) = \sum_{i=1}^{t} \omega_i(\boldsymbol{\pi})$ for an arbitrary ordering vector $\boldsymbol{\pi} \in \Pi(\boldsymbol{s})$. Suppose there exists a running sum maximizer $\boldsymbol{\pi}^*$ which satisfies $S_t(\boldsymbol{\pi}) \leq S_t(\boldsymbol{\pi}^*)$ for every $t \in \{1, \cdots, k\}$ and every $\boldsymbol{\pi} \in \Pi(\boldsymbol{s})$. Then, $\boldsymbol{\pi}^*$ turns out to be the min-cut minimizer. Showing that the vertical ordering vector $\boldsymbol{\pi}_v$ is the unique running sum maximizer completes the proof, which is omitted here.

Before stating our second main Lemma, here we define a special selection vector called the *horizontal selection vector*.

Definition 3. *The horizontal selection vector $\boldsymbol{s}_h = [s_1, \cdots, s_L] \in \mathcal{S}$ is defined as:*

$$s_i = \begin{cases} n_I, & i \leq \lfloor \frac{k}{n_I} \rfloor \\ k - \lfloor \frac{k}{n_I} \rfloor n_I, & i = \lfloor \frac{k}{n_I} \rfloor + 1 \\ 0 & i > \lfloor \frac{k}{n_I} \rfloor + 1. \end{cases}$$

The graphical illustration of the horizontal selection vector is on the left side of Fig. 8, in the case of $n = 15, k = 8, L = 3$. Now, our second main Lemma states that the horizontal selection vector minimizes the min-cut.

Lemma 2. *When the vertical ordering $\boldsymbol{\pi}_v$ is selected, the horizontal selection vector $\boldsymbol{s}_h$ minimizes the min-cut. In other words, $c_{min}(\boldsymbol{s}_h, \boldsymbol{\pi}_v) \leq c_{min}(\boldsymbol{s}, \boldsymbol{\pi}_v)$ $\forall \boldsymbol{s} \in \mathcal{S}$.*

The proof of this Lemma will be given elsewhere, but it is based on following logic. For the given horizontal selection vector $\boldsymbol{s}_h$ and the corresponding vertical ordering vector $\boldsymbol{\pi}_v$, denote the i^{th} weight value in (3) as ω_i^*. Similarly, for an arbitrary selection vector $\boldsymbol{s}$ and the corresponding vertical ordering vector $\boldsymbol{\pi}_v$, denote the i^{th} weight value as ω_i. Then, it can be shown that $\omega_i^* \leq \omega_i$ for $i = 1, \cdots, k$. Therefore, the horizontal selection minimizes the min-cut directly from (2).

C. Main Theorem: Capacity of Clustered DSS

Now, we state our main result in the form of a theorem which offers a closed-form solution for the capacity of the

clustered DSS. Note that setting $L = 1$ or $\beta_I = \beta_c$ reduces to the capacity of non-clustered DSS obtained in [5].

Theorem 1. *Consider a $\beta_I \geq \beta_c$ case. The capacity of the clustered distributed storage system with parameters $(n, k, L, \alpha, \gamma_I, \gamma_c)$ is*

$$\mathcal{C}(\alpha, \gamma_I, \gamma_c) = \sum_{i=1}^{n_I} \sum_{j=1}^{g(i)} \min\{\alpha, x(i)\gamma_I + y(i,j)\gamma_c\}, \quad (6)$$

where

$$x(i) = (n_I - i)/(n_I - 1)$$
$$y(i,j) = \{n - (n_I - i) - \sum_{m=1}^{i-1} g(m) - j\}/(n - n_I)$$
$$g(i) = \begin{cases} \lfloor \frac{k}{n_I} \rfloor + 1, & i \leq mod(k, n_I) \\ \lfloor \frac{k}{n_I} \rfloor, & otherwise. \end{cases}$$

Proof. From Lemmas 1 and 2, we have

$$\forall \boldsymbol{s} \in \mathcal{S}, \forall \boldsymbol{\pi} \in \Pi(\boldsymbol{s}),\ c_{min}(\boldsymbol{s}_h, \boldsymbol{\pi}_v) \leq c_{min}(\boldsymbol{s}, \boldsymbol{\pi}). \quad (7)$$

Therefore, from the explanation on G^* in the beginning of Section IV, the min-cut minimizing flow graph G^* is generated by selecting k output nodes $\{x_{out}^{c_i}\}_{i=1}^{k}$ by the *horizontal selection vector* $\boldsymbol{s}_h$ and ordering these nodes by the *vertical ordering* $\boldsymbol{\pi}_v$, as illustrated in Fig. 8. The min-cut value of G^*, or $c_{min}(\boldsymbol{s}_h, \boldsymbol{\pi}_v)$, is summarized as (6). □

D. Relationship between $\mathcal{C}$ and $\kappa = \beta_c/\beta_I$

In this subsection, we analyze the capacity of a clustered DSS as a function of a new important parameter $\kappa = \beta_c/\beta_I$: the cross-cluster repair burden compared to the intra-cluster repair burden. In Fig. 9, the capacity is plotted as a function of κ. The total repair bandwidth can be expressed as

$$\begin{aligned} \gamma = \gamma_I + \gamma_c &= (n_I - 1)\beta_I + (n - n_I)\beta_c \\ &= \{n_I - 1 + (n - n_I)\kappa\}\beta_I. \end{aligned} \quad (8)$$

Using this expression, the capacity is expressed as

$$\mathcal{C}(\kappa) = \sum_{i=1}^{n_I} \sum_{j=1}^{g(i)} \min\{\alpha, \frac{(n - n_I)y(i,j)\kappa + (n_I - 1)x(i)}{(n - n_I)\kappa + n_I - 1}\gamma\}.$$

For fair comparison on various κ values, $(n, k, L, \alpha, \gamma)$ values are fixed for calculating the capacity. The capacity shows an increasing function of κ as in Fig. 9. This implies that for given resources α and γ, allowing a larger β_c (until it reaches β_I) is always beneficial, in terms of storing a larger file. For example, under the setting in Fig. 9, allowing $\beta_c = \beta_I$ (i.e., $\kappa = 1$) can store $\mathcal{M} = 48$, while setting $\beta_c = 0$ (i.e., $\kappa = 0$) cannot achieve the same level of storage. This result is consistent with the previous work on asymmetric repair in [11], which proved that the symmetric repair maximized the capacity. Therefore, when the total communication amount γ is fixed, the reduction of the storage capacity is the cost we need to pay in order to reduce the communication burden β_c across different clusters.

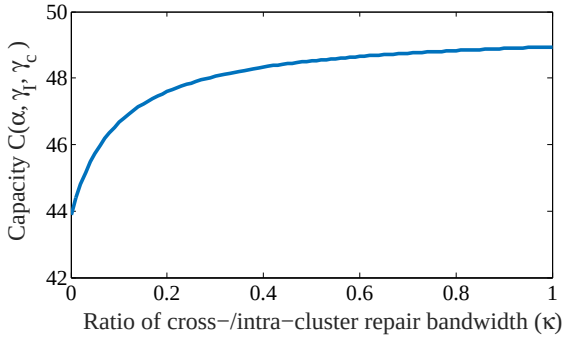

Fig. 9: Capacity as a function of κ ($n = 100, k = 85, L = 10, \alpha = 1, \gamma = 1$)

V. Discussion on Feasible $(\alpha, \gamma_I, \gamma_c)$

In the previous section, we obtained the capacity of the clustered DSS. This section analyzes the feasible $(\alpha, \gamma_I, \gamma_c)$ points which satisfy $\mathcal{C}(\alpha, \gamma_I, \gamma_c) \geq \mathcal{M}$ for a given file size $\mathcal{M}$. Here, the behavior of feasible points are analyzed in two different perspectives. First, we focus on the $\gamma_c = 0$ case, which enables the local repair within each cluster. Second, we analyze the trade-off relation between γ_I and γ_c.

A. Intra-Cluster Repairable Condition ($\gamma_c = 0$)

Under the constraint of zero cross-cluster repair bandwidth, a closed-form solution for the feasible points (α, γ_I) can be obtained as the following corollary. This corollary can be proved by substituting $\gamma_c = 0$ to (6), and obtaining an equivalent condition for (α, γ_I) which satisfies $\mathcal{C}(\alpha, \gamma_I, 0) \geq \mathcal{M}$. A detailed proof will be given elsewhere.

Corollary 1. *Consider a clustered DSS for storing data $\mathcal{M}$. Then, for any $\gamma_I \geq \gamma_I^*(\alpha)$, the cross-cluster repair bandwidth can be reduced to zero, i.e., $\mathcal{C}(\alpha, \gamma_I, 0) \geq \mathcal{M}$, while it is impossible to reduce cross-cluster repair bandwidth to zero when $\gamma_I < \gamma_I^*(\alpha)$. The threshold function $\gamma_I^*(\alpha)$ can be obtained as:*

$$\gamma_I^*(\alpha) = \begin{cases} \infty, & \alpha \in (0, \frac{M}{\frac{\epsilon_{n_I-2}}{b_{n_I-2}} + \delta_{n_I-2}}] \\ \frac{M - \delta_t \alpha}{\epsilon_t}, & \alpha \in [\frac{M}{\epsilon_t/b_t + \delta_t}, \frac{M}{\epsilon_{t-1}/b_{t-1} + \delta_{t-1}}), \\ & 1 \leq t \leq n_I - 2 \\ \frac{M}{\epsilon_0}, & \alpha \in (\frac{M}{\epsilon_0}, \infty) \end{cases}$$

where

$$\epsilon_t \triangleq \lfloor \frac{k}{n_I} \rfloor \sum_{i=t}^{n_I - 1} b_i + \sum_{i=t}^{mod(k, n_I) - 1} b_i$$
$$\delta_t \triangleq \begin{cases} (\lfloor \frac{k}{n_I} \rfloor + 1)t, & 0 \leq t \leq mod(k, n_I) \\ \lfloor \frac{k}{n_I} \rfloor t + mod(k, n_I), & mod(k, n_I) < t \leq n_I - 1 \end{cases}$$
$$b_t \triangleq 1 - t/(n_I - 1).$$

An example for the tradeoff result of Corollary 1 is illustrated in Fig. 10. The plot for the $\beta_I = \beta_c$ case is illustrated as the dotted line. The solid line represents the $\gamma_c = 0$ case,

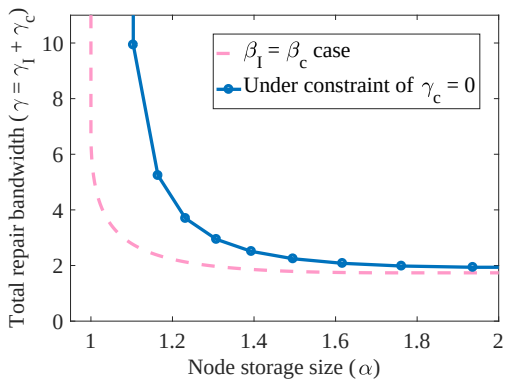

Fig. 10: Optimal tradeoff between node storage size α and total repair bandwidth γ ($n = 100, k = 85, L = 10, \mathcal{M} = 85$)

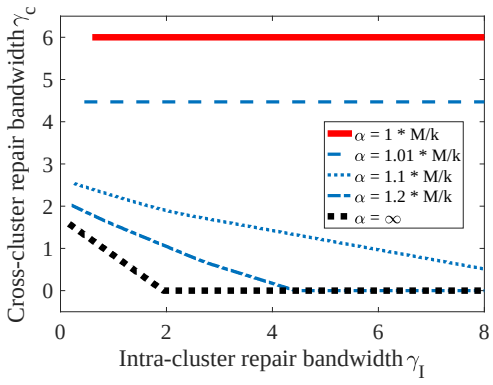

Fig. 11: Optimal tradeoff between intra-/cross-cluster repair bandwidth ($n = 100, k = 85, L = 10, \mathcal{M} = 85$)

where in this case $\gamma = \gamma_I$. This shows that the cross-cluster repair bandwidth can be reduced to zero with extra resources (α and γ_I), where the set of feasible pairs of these resources is specified in Corollary 1. Note that $\gamma_c = 0$ means that the *local repair in each cluster is possible*.

B. Trade-off between γ_I and γ_c

In this subsection, we focus on the following problem: Given α, what is the feasible set of (γ_I, γ_c) pairs which satisfy $\mathcal{C}(\alpha, \gamma_I, \gamma_c) = \mathcal{M}$? The sets of feasible (γ_I, γ_c) pairs for various α values is illustrated in Fig. 11. Note that the plots in the figure show a decreasing function of α for a given γ_I. This has been obtained from the capacity expression of (6).

For small α values ($\mathcal{M}/k$ or $1.01 \times \mathcal{M}/k$ in the figure), the cross-cluster repair bandwidth cannot be reduced irrespective of the intra-cluster repair bandwidth γ_I. For sufficiently large α values, trade-off between γ_c and γ_I can be observed, with γ_c settling to zero for large γ_Is as α increases.

VI. Conclusion

This paper considered a practical distributed storage system where storage nodes are dispersed into several clusters. Noticing that the traffic burdens of intra- and cross-cluster communication are different, a new system model for a clustered distributed storage system is suggested. Based on the min-cut analysis of information flow graph, the storage capacity of the suggested model is obtained in a closed-form, as a function of three main resources: node storage size α, intra-cluster repair bandwidth γ_I and cross-cluster repair bandwidth γ_c. It is shown that the asymmetric repair ($\beta_I > \beta_c$) degrades the capacity, which is the cost to pay for reducing cross-cluster repair burden. Moreover, local repair ($\gamma_c = 0$) is shown to be possible at the expense of extra resources (α and γ_I), where the amounts of required extra resources are specified in a mathematical form. Finally, the feasible (γ_I, γ_c) pairs showed a clear trade-off relation for large enough storage size α.

References

[1] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *ACM SIGOPS operating systems review*, vol. 37, no. 5. ACM, 2003, pp. 29–43.

[2] R. Bhagwan, K. Tati, Y. Cheng, S. Savage, and G. M. Voelker, "Total recall: System support for automated availability management." in *NSDI*, vol. 4, 2004, pp. 25–25.

[3] F. Dabek, J. Li, E. Sit, J. Robertson, M. F. Kaashoek, and R. Morris, "Designing a dht for low latency and high throughput." in *NSDI*, vol. 4, 2004, pp. 85–98.

[4] S. C. Rhea, P. R. Eaton, D. Geels, H. Weatherspoon, B. Y. Zhao, and J. Kubiatowicz, "Pond: The oceanstore prototype." in *FAST*, vol. 3, 2003, pp. 1–14.

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

[6] R. Ahlswede, N. Cai, S.-Y. Li, and R. W. Yeung, "Network information flow," *IEEE Transactions on information theory*, vol. 46, no. 4, pp. 1204–1216, 2000.

[7] D. Ford, F. Labelle, F. I. Popovici, M. Stokely, V.-A. Truong, L. Barroso, C. Grimes, and S. Quinlan, "Availability in globally distributed storage systems." in *OSDI*, 2010, pp. 61–74.

[8] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin, "Erasure coding in windows azure storage," in *Presented as part of the 2012 USENIX Annual Technical Conference (USENIX ATC 12)*, 2012, pp. 15–26.

[9] S. Muralidhar, W. Lloyd, S. Roy, C. Hill, E. Lin, W. Liu, S. Pan, S. Shankar, V. Sivakumar, L. Tang *et al.*, "f4: Facebook's warm blob storage system," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, 2014, pp. 383–398.

[10] F. Ahmad, S. T. Chakradhar, A. Raghunathan, and T. Vijaykumar, "Shufflewatcher: Shuffle-aware scheduling in multi-tenant mapreduce clusters," in *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, 2014, pp. 1–13.

[11] T. Ernvall, S. El Rouayheb, C. Hollanti, and H. V. Poor, "Capacity and security of heterogeneous distributed storage systems," *IEEE Journal on Selected Areas in Communications*, vol. 31, no. 12, pp. 2701–2709, 2013.

[12] S. Akhlaghi, A. Kiani, and M. R. Ghanavati, "A fundamental trade-off between the download cost and repair bandwidth in distributed storage systems," in *2010 IEEE International Symposium on Network Coding (NetCod)*. IEEE, 2010, pp. 1–6.

[13] B. Gastón, J. Pujol, and M. Villanueva, "A realistic distributed storage system that minimizes data storage and repair bandwidth," *arXiv preprint arXiv:1301.1549*, 2013.

[14] M. A. Tebbi, T. H. Chan, and C. W. Sung, "A code design framework for multi-rack distributed storage," in *Information Theory Workshop (ITW), 2014 IEEE*. IEEE, 2014, pp. 55–59.

[15] Y. Hu, P. P. C. Lee, and X. Zhang, "Double regenerating codes for hierarchical data centers," in *2016 IEEE International Symposium on Information Theory (ISIT)*, July 2016, pp. 245–249.

[16] A. G. Dimakis, K. Ramchandran, Y. Wu, and C. Suh, "A survey on network codes for distributed storage," *Proceedings of the IEEE*, vol. 99, no. 3, pp. 476–489, 2011.

[17] J. Bang-Jensen and G. Z. Gutin, *Digraphs: theory, algorithms and applications*. Springer Science & Business Media, 2008.