Normalized Comparison Method for Finding the Most Efficient DSS Code

Natasa Paunkoska (Dimoska)[1(✉)], Ninoslav Marina[1], and Weiler Finamore[2]

[1] University of Information Science and Technology (UIST) "St. Paul the Apostle", Ohrid, North Macedonia
natasa.paunkoska@uist.edu.mk, ninoslav.marina@gmail.com
[2] Federal University of Juiz de Fora (UFJF), Juiz de Fora, Brazil
weilerfinamore44@gmail.com

Abstract. Big data is large volume of data produced on a daily basis. Distributed storage systems (DSS) is environment that handles, manages and stores those data. The main drawbacks are the lack of system storage capacity, the network device failures that can appear anytime, on time data processing and the system efficiency. All above mentioned issues can be overcome by applying different coding techniques for data distribution. Till this moment many coding schemes are proposed by the researchers. Determining the most efficient code for usage is still a tricky question, which yields an adequate comparison strategy for code selection. The basic Dimakis comparison method offers analysis between the codes regarding the parameters storage per node and download bandwidth to be repair one node. Total comparison method includes in the analysis the total number of nodes in the system together with the overall storage and total downloaded bandwidth in the repair process, with notation that the file size for all codes must be same. In this paper, we are proposing new method for comparison, called Normalized, that enables consideration of broader spectrum of parameters and not necessarily the same file size of the proposed codes.

Keywords: Comparison methods · Distributed storage systems (DSS) · Efficient DSS system · Reconstruction process · Repair process

1 Introduction

Cloud data centers are very important thread for data management, especially of the 'Big Data' production. These centers acts as distributed storage systems (DSS). DSS is network of many server (nodes) interconnected among them that stores large number of data, keeps the data reliable and safe, and enables fast retrieval of the message for the user. The main function, data storage, is performed as the information (file), of size $|\mathcal{F}|$ bytes, is divided into smaller pieces and then each piece is distributed on physically separated device (node) in the network. The possibility for network/device failure that guarantee data reliability is provided by adding redundancy. Currently, most of the systems use

R. Zitouni et al. (Eds.): AFRICOMM 2020, LNICST 361, pp. 253–268, 2021.
https://doi.org/10.1007/978-3-030-70572-5_16

repetition code for redundancy, i.e, two more copies of each file piece are stored in different locations (nodes). The consequence of this technique is the storage overhead.

Introducing the Maximum Distance Separable (MDS) codes adapted for DSS [1–4] deals the information storage problem by making optimal storage vs. reliability data trade-off. Erasure codes are also interesting for the researchers [5–10]. They are efficient in decreasing the overall system storage. The codes function in a manner that they are taking the entire message $\mathcal{F}$ of size $|\mathcal{F}| = LK$ and parses in blocks of size K. L is number of info-words and K number of symbols within one info-word to be distributed/stored in the DSS system. Then each block K is divided into smaller chunks, with size α, and each chunk is stored in one of the n nodes in the network. The chunk is calculated as $\alpha = \frac{K}{k}$, where k is number of info symbols for the encoding/storage process. Knowing that the DSS system is defined as network of n nodes (servers), the parameters n and k depends of the code $[n, k]$ chosen for data storage. Later, the value $k(k < n)$ is used for determining the number of nodes to which the Data collector (DC) connects in order to retrieve the original stored information. This process in a sense of DSS is known as *reconstruction*.

Another issue in the DSS network is a node failure and losing everything stored on it. In this situation if node i fails, a new node i' (newcomer) that will replace the failed one is programmed to connect to any k surviving nodes, to download from them everything what is stored, by α amount of data, and by proper computation to restore the lost information. This process is known as *repair*. For completing of this process, a lot of unnecessary data going to be downloaded and just few of them will be used in the computations. This means inefficiently high download repair bandwidth that the erasure codes can not handle with.

Interesting solution that deals this problem is introduced by Dimakis et al. [11,12]. There, the authors offer new kind of codes known as regenerating. The code characterization is done by the primary parameters $[n, k, d]$ and secondary parameters $[\alpha, \beta, K]$. The meaning of n, k and α is same as the one mentioned above, $d(d \geq k)$ shows the number of nodes that need to be contacted to be finish the repair process and $\beta(\beta < \alpha)$ is the amount of data downloaded from those d nodes. Precisely, during the *repair process*, a newcomer contacts d alive nodes and downloads from them β quantity of information, unlike in erasure codes, where contacts k nodes and downloads from there everything what is stored α. Thus, the *repair bandwidth* is $\gamma = d\beta$ symbols and is used to recover what was lost. Some researches that are based on the regenerating codes can be found in [13–17].

Till now, there are lot of code constructions proposed that will work efficiently in DSSs. How to choose which one performs the best, is an open question in this area. The simplest solution is to compare the codes. But, finding a suitable comparison method to do that is a problem. The first attempt is made by Dimakis et al. in [12]. Their method use two parameters for comparison, either will be measured by the storage per node, α, or repair bandwidth for one node, $d\beta$.

Later, in [18] is introduced the Total method that extends the comparison to a total system storage and a total repair bandwidth for more failed nodes. In this paper, we are proposing new method for comparison, called Normalized, that gives more realistic output than the existing ones. The proposal is upgrade of the Total concept and the normalization is done over the file size. This allows codes comparison with different file size, in contrast to the others where the file size must be the same. The validity and efficiency of our approach is done through examples, where we compare four various codes (Regenerating, Repetition, Reed-Solomon based DSS and Clay) using the three above mentioned methods.

The paper is organized as follows, in Sect. 2 are given the theorems that describe the trade-off curves of the Dimakis, Total and the new proposed comparison method. In Sect. 3 is given description of the codes involved in the comparison procedures. Section 4 analysis the results from the comparison processes and Sect. 5 concludes the paper.

2 Trade-Off Curve for DSS

Coding for DSS is a process that first parses the input-string $\mathcal{F}$, of size $|\mathcal{F}|$, into L info-words $\underline{u}^{[\ell]}$ each K symbols (q-ary symbols) long. Next, the data from each info-word $\underline{u}$ (focusing on one block, thus drop the index ℓ) is mapped into a code-matrix $\underline{C}$ of dimensions $n \times \alpha$,

$$\underline{C} = \begin{pmatrix} \underline{C}_1 \\ \vdots \\ \underline{C}_i \\ \vdots \\ \underline{C}_n \end{pmatrix}, \tag{1}$$

where the content of each $\underline{C}$ row is stored in one of the n nodes in the network.

In general the encoder is a vector function, $\underline{\mathcal{E}} = (\underline{\mathcal{E}}_1 \ldots \underline{\mathcal{E}}_i \ldots \underline{\mathcal{E}}_n)$ which yields the code-matrix $\underline{C} = \underline{\mathcal{E}}(\underline{u})$. Thus, $R = \frac{k}{\alpha n}$ is the code rate. The reconstruction process happens when the data collector (DC) wants to reconstruct the original information $\mathcal{F}$ stored in the system. During this process, the DC connects to any $k(n > k \leq d)$ nodes and downloads from them by α quantity of symbols. Overall collects $k\alpha$ data and from them obtains the entire original message of size $|\mathcal{F}| = LK \log_2 q$ bytes. Mathematically, the reconstruction process can be described as

$$\underline{u} = \underline{\mathcal{D}}(\underline{C}_{i_1} \ldots \underline{C}_{i_\ell} \ldots \underline{C}_{i_k}).$$

The repair process happens when a node fails, and the trigger is prompted by a new node (newcomer), which connects to $d(n-1 \geq d \geq k)$ alive nodes in the network and downloads from them by $\beta(\beta \leq \alpha)$ symbols. The total downloaded bandwidth is $\gamma = \beta d$ symbols that is used to recover the lost data previously stored in the failed node. In other words

$$\underline{u} = \mathcal{R}(\underline{C}_{j_1} \ldots \underline{C}_{j_\ell} \ldots \underline{C}_{j_d}).$$

Authors in [11] proved that code with parameters (k, d, α, β) of a q-ary regenerating code that reliably stores one info-word of size K must satisfy the following condition

$$K \leq \sum_{i=0}^{k-1} \min\left\{\alpha, (d-i)\,\beta\right\}. \tag{2}$$

The download-bandwidth and the amount of storage can not be minimized in a same time, therefore, a tradeoff between the repair bandwidth, $\gamma = d\beta$, and the storage per node, α, exists and is made explicit in Theorem 1.

Theorem 1 (Storage-Bandwidth Tradeoff curve [11]). *For any value $\alpha \geq \alpha^*(n, k, d, \gamma)$, the points $(n, k, d, \alpha, \gamma)$ are feasible, and linear network codes suffice to achieve them. It is information theoretically impossible to achieve points with $\alpha < \alpha^*(n, k, d, \gamma)$. The threshold function $\alpha^*(n, k, d, \gamma)$ is the following:*

$$\alpha^*(n,k,d,\gamma) = \begin{cases} \frac{K}{k} & \gamma \in [f(0), +\infty] \\ \frac{K-g(i)\gamma}{k-i} & \gamma \in [f(i), +f(i-1)] \end{cases} \tag{3}$$

where

$$f(i) \triangleq \frac{2Kd}{(2k-i-1)i + 2k(d-k+1)}, \tag{4}$$

$$g(i) \triangleq \frac{(2d-2k+i+1)i}{2d}. \tag{5}$$

where $d \leq n-1$. ♦

The storage-bandwidth tradeoff theorem in [11], here refer as Dimakis, establishes that no code exists with pairs of values that are under this curve.

In [18] authors made modification of the previous Dimakis trade-off curve and relates the total storage in the system and the total bandwidth needed for the repair process. Definition of this trade-off curve is given in following.

Theorem 2 (Total Comparison [18]). *Let $(n, K, k, d, \alpha, \beta)$, when $d \leq n-1$, be a set of parameters for a DSS code. The Total-downloaded-data and the Total-storage-data trade-off curve $S^{total}(B^{total})$ joining the points $(B_i^{total}, S_i^{total})$ $i \in \{0, \ldots, k-1\}$ is given by*

$$B_i^{total} = \frac{2dKL}{(2k-i-1)i + 2k(d-k+1)}, \tag{6}$$

$$S_i^{total} = \frac{n}{k-i}(KL - g_i' B_i^{total}), \tag{7}$$

in which

$$g_i' = \frac{(2d-2k+i+1)i}{2d}, \tag{8}$$

is, by definition, the Total Storage-Bandwidth Trade Off Curve *for codes with the given set of parameters.* ♦

Proof. The Total comparison extends the Theorem 1 idea introducing wider picture of the DSS system. Precisely, takes care not only on one-node repair bandwidth, $\gamma = d\beta$, and storage per node, α, but also counts the number of nodes in the DSS and potentially the necessity of having more nodes for repairing. Thus, modifying Eq. (4), (3) and (5) by multiplying with L and Ln, accordingly, are obtained (6) and (7).

Theorem 3 (Total Storage-Bandwidth Feasible Region). *A DSS code with parameters* $(n, K, k, d, \alpha, \beta)$, $d \leq n-1$, *has Total Bandwidth,* $t_\beta = \beta dL \log_2 q$, *and Total Storage,* $t_\alpha = \alpha nL \log_2 q$, *such that*

$$t_\alpha \geq S^{total}(t_\beta) \quad \text{if} \quad B_{k-1}^{total} \leq t_\beta \leq B_0^{total} \tag{9}$$

$$t_\alpha \geq S^{total}(B_0^{total}) \quad \text{if} \quad t_\beta \geq B_0^{total} \tag{10}$$

no DSS code exists with $t_\beta < B_{k-1}^{total}$. ♦

Proof. One node repair bandwidth is defined as number of nodes, d, contacted during the repair process of one failed node and downloaded from them by β quantity of information, or $\gamma = d\beta$. Storage per node, α, is the amount of data stored on each node in the DSS. The Total concept considers also all nodes in the system, the info-word, file size and all potential simultaneously failed nodes. So, calculating now the overall DSS storage the obtained Total Storage is $\alpha nL \log_2 q$, and the whole downloaded bandwidth or Total bandwidth becomes $\beta dL \log_2 q$. The total storage-bandwidth tradeoff theorem establishes that no code exists with pairs of values (t_α, t_γ) that are under the curve S^{total} versus B^{total}, defined with Theorem 2. Because those two values are interdependent, means that the Total Storage value must be above each point that lies on the curve with correspondence to the Total Bandwidth.

In this paper, the main contribution is introduction of new comparison method. The proposed method is a variation of both the storage-bandwidth tradeoff theorem and total storage-total downloaded data tradeoff theorems. A similar tradeoff curve is established relating the quantities Normalized Storage and Normalized Download-bandwidth, defined as follows.

Definition 1 (Normalized Storage). *Let* $\mathcal{F}$ *be a file to be stored in a Distributed Storage System. The ratio of the total number of bytes stored on* n *nodes of the network,* $\alpha nL \log_2 q$, *to the size of the file,* $|\mathcal{F}| = LK \log_2 q$ *(bytes), or, in other words,*

$$\eta_\alpha \triangleq \frac{\alpha nL \log_2 q}{LK \log_2 q} = \frac{\alpha n}{K} \tag{11}$$

is defined as Normalized Storage. ♦

Proof. Normalizing the Total Storage, $t_\alpha = \alpha nL \log_2 q$, over the file size, $|\mathcal{F}| = LK \log_2 q$, is obtained Eq. (11).

Definition 2 (Normalized Download-bandwidth). *Let $\mathcal{F}$ be a file to be stored in a Distributed Storage System. The ratio of the total number of bytes downloaded from d nodes of the network (when the procedure to repair one node out of the n nodes is activated), $\beta dL \log_2 q$, to the size of the file, $|\mathcal{F}| = LK \log_2 q$ (bytes), or, in other words,*

$$\eta_\beta \triangleq \frac{\beta dL \log_2 q}{LK \log_2 q} = \frac{\beta d}{K} \tag{12}$$

is defined as Normalized Download-bandwidth. ♦

Proof. Normalizing the Total Bandwith, $t_\beta = \beta dL \log_2 q$, over the file size, $|\mathcal{F}| = LK \log_2 q$, is obtained Eq. (12).

The trade-off function relating S^{nor} (the normalized storage) and B^{nor} (the normalized bandwidth) is described by the theorem stated next.

Theorem 4 (Normalized Storage-Bandwidth Tradeoff curve). *Let a set of parameters for a DSS code be $(n, K, k, d, \alpha, \beta)$, when $d \leq n-1$. The piecewise linear curve $S^{nor}(B^{nor})$ joining the points (B_i^{nor}, S_i^{nor}) $i \in \{0, \ldots, k-1\}$ is given by*

$$B_i^{nor} = \frac{2d}{(2k-i-1)i + 2k(d-k+1)}, \tag{13}$$

$$S_i^{nor} = \frac{n}{k-i}(1 - g_i' B_i^{nor}), \tag{14}$$

in which

$$g_i' = \frac{(2d-2k+i+1)i}{2d} \tag{15}$$

is, by definition, the Normalized Storage-Bandwidth Trade Off Curve *for codes with the given set of parameters.* ♦

Proof. The Normalized comparison extends the Theorem 2 idea by normalizing the outcomes with the number of symbols stored in the system, K, derived form Eq. (11) and (12). Precisely, normalizing Eq. (6) with K is obtained (13) and normalizing Eq. (7) with K is obtained (14).

Theorem 5 (Normalized Storage-Bandwidth Feasible Region). *A DSS code with parameters $(n, K, k, d, \alpha, \beta)$, $d \leq n-1$, has Normalized Bandwidth, η_β, and Normalized Storage, η_α, such that*

$$\eta_\alpha \geq S^{nor}(\eta_\beta) \quad \text{if} \quad B_{k-1}^{nor} \leq \eta_\beta \leq B_0^{nor} \tag{16}$$

$$\eta_\alpha \geq S^{nor}(B_0^{nor}) \quad \text{if} \quad \eta_\beta \geq B_0^{nor} \tag{17}$$

no DSS code exists with $\eta_\beta < B_{k-1}^{nor}$. ♦

Proof. Normalized Storage is defined with Eq. (11) and Normalized Bandwidth with (12). The normalized storage-bandwidth tradeoff theorem establishes that no code exists with pairs of values $(\eta_\alpha, \eta_\gamma)$ that are under the curve S^{nor} versus B^{nor}, defined with Theorem 4. Because those two values are interdependent, means that the Normalized Storage value must be above each point that lies on the curve with correspondence to the Normalized Bandwidth.

The main motivation behind Theorem 5 is its ability to allow comparison of diverse DSS coding schemes (having disparate parameters). Let say, if we compare two codes by the storage per node parameter, then the code with smaller storage is better. This approach have unfair output, because for example doesn't care about the overall system storage and in this sense may give worse performance. Hence, the coding scheme comparison based on the normalized-storage versus normalized-bandwidth tradeoff curves is file size independent, unlike Dimakis and Total methods described in Theorem 1 and Theorem 2. To make this point clear we examine and illustrate the quality of the tradeoff curves using three diverse codes. The chosen codes are defined with values as: first one $(n, K, k, d) = (14, 1, 8, 13)$, second $(n, K, k, d) = (10, 1, 6, 9)$ and third $(n, K, k, d) = (15, 2, 10, 14)$. Figure 1 is based on Theorem 1, Fig. 2 on Theorem 2 and Fig. 3 on Theorem 4. The plots give the trade off curves applied on the chosen codes together with their achievable regions.

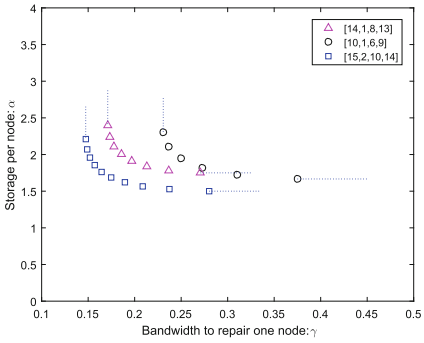

Fig. 1. Trade-off curve Storage-per-node vs. Total-repair-bandwidth for parameters $(n, K, k, d) = (14, 1, 8, 13)$ (marked with a triangle), $(n, K, k, d) = (10, 1, 6, 9)$ (marked with a circle) and $(n, K, k, d) = (15, 2, 10, 14)$ (marked with a square).

From the graphs we can conclude that all three comparison methods gives different outputs. Dimakis method take into consideration only two parameters: storage per node and bandwidth to repair one node and claims that square code is the best. Total method includes plus the number of nodes in the system yielding the overall storage and total downloaded bandwidth for the repair process and favors the circle code. And the normalized concept gives broader picture of parameters by allowing choosing arbitrary code file size and demonstrate that the triangle code is best.

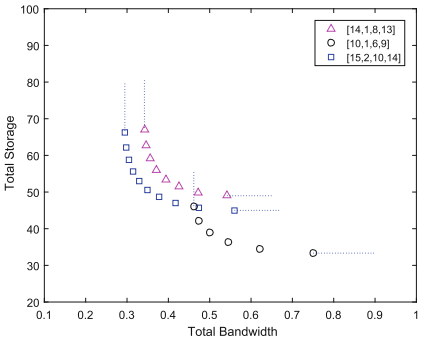

Fig. 2. Trade-off curve Total storage vs. Total repair bandwidt considering fair comparison procedure for parameters $(n, K, k, d) = (14, 1, 8, 13)$ (marked with a triangle), $(n, K, k, d) = (10, 1, 6, 9)$ (marked with a circle) and $(n, K, k, d) = (15, 2, 10, 14)$ (marked with a square).

3 Overview of DSS Codes

This section elaborates the construction of four diverse DSS codes that will be used in the comparison analysis. Precisely, gives description for the functioning and efficiency of the Regenerating Code, Repetition, Reed-Solomon based DSS and Clay code.

3.1 Regenerating Code

The general Regenerating code construction is presented in [11]. Here, we will give explicit construction through an example.

Example 1. The parameters of the code $\mathcal{C}[n, k, d]$, are $n = 4$, $k = 2$, $d = 3$. The file size $|\mathcal{F}|$ is arranged as a vector $\underline{u} = (u_1, \ldots, u_K)$ of $K = k\alpha = 4$ q-ary symbols, where $\alpha = 2$. Further, the vector $\underline{u}$ is arranged in a matrix M given as,

$$M = \begin{pmatrix} u_1 & u_2 \\ u_3 & u_4 \end{pmatrix}. \tag{18}$$

Then the matrix M is multiply by encoding matrix Ψ with properties explained in [11] and a code-matrix is generated,

$$\underline{C} = \begin{pmatrix} u_1 & u_2 \\ u_3 & u_4 \\ u_1 + u_2 + u_3 + u_4 & u_1 + 2u_2 + u_3 + 2u_4 \\ u_1 + 2u_2 + 3u_3 + u_4 & 3u_1 + 2u_2 + 3u_3 + 3u_4 \end{pmatrix}, \tag{19}$$

with dimension $(\alpha \times n) = (2 \times 4)$. The number of distinct blocks or vectors $\underline{u} = (u_1\ u_2\ u_3\ u_4)$ in $\mathcal{F}$ is L, then, $|\mathcal{F}| = LK \log_2 q$. In this example, for $|\mathcal{F}| = 4$ symbols, means $L = 1$.

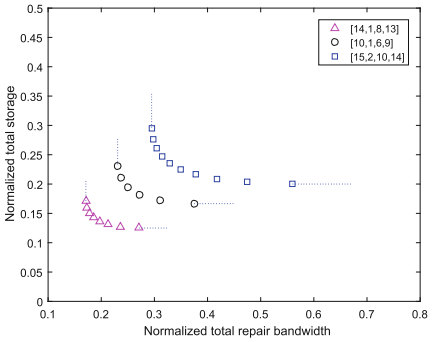

Fig. 3. Trade-off curve Normalized total storage vs. Normalized total repair bandwidth for parameters $(n, K, k, d) = (14, 1, 8, 13)$ (marked with a triangle), $(n, K, k, d) = (10, 1, 6, 9)$ (marked with a circle) and $(n, K, k, d) = (15, 2, 10, 14)$ (marked with a square).

The reconstruction of the original message is done by downloading $\alpha = 2$ data from any $k = 2$ nodes (choosing any two rows from the code-matrix $\underline{C}$). The vector $\underline{u} = (u_1\ u_2\ u_3\ u_4)$ then is obtained by solving four equations with four unknowns.

For the repair process if node $i = 4$ fail, the new added node, $i' = 5$ (the repair-node), contacts $d = 3$ nodes and downloads from them by $\beta = 1$ data. The collected data is represented by the parity-information $\underline{P}$ matrix of dimension $d \times \beta$ given as

$$\begin{aligned} \underline{P} &= \begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix} = \begin{pmatrix} c_{1,1} + 2c_{1,2} \\ 2c_{2,1} + c_{2,2} \\ 3c_{3,1} + c_{3,2} \end{pmatrix} \\ &= \begin{pmatrix} u_1 + 2u_2 \\ 2u_3 + u_4 \\ 4u_1 + 5u_2 + 4u_3 + 5u_4 \end{pmatrix}. \end{aligned} \tag{20}$$

With these information the newcomer regenerates the lost vector by proper combination of the parity vector components. In this example the regenerated vector would be

$$(c_{5,1}\ c_{5,2}) = (5u_1 + 7u_2 + 8u_3 + 7u_4,\ 6u_1 + 9u_2 + 6u_3 + 6u_4),$$

obtained with multiplication among $\underline{P}$ and

$$\begin{pmatrix} 1 & 2 \\ 2 & 1 \\ 1 & 1 \end{pmatrix}. \tag{21}$$

Thus, the newly stored repaired information in the network would be

$$\underline{C} = \begin{pmatrix} u_1 & u_2 \\ u_3 & u_4 \\ u_1+u_2+u_3+u_4 & u_1+2u_2+u_3+2u_4 \\ 5u_1+7u_2+8u_3+7u_4 & 6u_1+9u_2+6u_3+6u_4 \end{pmatrix}. \tag{22}$$

3.2 Repetition Code

Repetition code construction given in [5] takes the original message to be stored in the system and represents as a vector $\underline{u} = (u_1\ u_2\ \ldots\ u_{LK})$. What is distributed in the system is depicted by the matrix $\underline{C} = (\underline{C}^{[1]}; \ldots; \underline{C}^{[n]})$, where $\underline{C}^{[n]} = \ldots = \underline{C}^{[1]} = \underline{M} = \underline{u}$ is seen as multiplication of $1 \times n$ matrix with all elements equal to one's and the input vector $\underline{u}$, i.e.

$$\underline{C} = \begin{pmatrix} 1 \ldots 1 \end{pmatrix} \underline{M}. \tag{23}$$

Example 2. To create a repetition example we take $n = 3$, $k = 1$. Knowing that $d = k$, we have $d = 1$, and $n = \alpha = \beta$, $\alpha = \beta = 3$. For $K = 1$, $L = 1$, and message matrix $M = u_i$ for $i = 1, \ldots, n$ we get

$$\begin{aligned} \underline{C}^{[i]} &= \begin{pmatrix} 1\,1\,1 \end{pmatrix} M \\ &= \begin{pmatrix} u_i \\ u_i \\ u_i \end{pmatrix}. \end{aligned} \tag{24}$$

The reconstruction and repair processes are simply done by contacting k or d nodes, accordingly, and downloading from them by α or β data.

3.3 Reed-Solomon Based DSS Code

Reed-Solomon (RS) based DSS codes introduced in [4], which are MDS codes, are better alternative then the repetition codes. Each input block is a vector of k q-ary symbols. Hence, the elements of the code-matrix are belonging to a Finite Field $\mathbb{F}_q$. The file of size $|\mathcal{F}^{RS}| = K^{RS}L^{RS}$ is stored by parsing into L^{RS} vectors of size $k = K^{RS}$, then, they are encoded into a vector of size αn. For *reconstruction* of the original file, the DC contacts k distinct nodes and downloads from them k vectors of size $\beta = \alpha$ data. Next, it performs proper data decoding and obtains the wanted file. For the *repair process* new node contacts $k = d$ vectors of size α and downloads from them $\beta = \alpha$. Then, performs proper decoding and encoding, and the content of the failed node is regenerated.

Example 3. A RS based DSS-code $C[n, k, d]$ with parameters $[7, 3, 3]$ is obtained by taking the vector $\underline{u} = (u_1, u_2, \ldots, u_k)$ and using a RS-code generating matrix,

$$G = \begin{pmatrix} g_{1,1} & g_{1,2} & \cdots & g_{1,n} \\ g_{2,1} & g_{2,2} & \cdots & g_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ g_{k,1} & g_{k,2} & \cdots & g_{k,n} \end{pmatrix}. \tag{25}$$

Every block $\underline{u}$ will give rise to a RS-codeword $\underline{w} = (w_1, \ldots, w_j \ldots, w_n)$. With $K = k = 3$, $\beta = \alpha = 1$ and $d = k = 3$ we have the code-matrix

$$\underline{C} = \begin{pmatrix} w_1 \\ \vdots \\ w_\ell \\ \vdots \\ w_n \end{pmatrix}. \tag{26}$$

3.4 Clay Code

Clay (short for CoupledLayer) codes represents a subset of the MSR (Minimum Storage Regenerating) codes. They have simpler construction for the decoding repair process, achieved by pairwise coupling across multiple stacked layers of any single MDS code [19]. Using this way of code construction it can be decreased the normalized repair bandwidth by increasing the value of $(d - k + 1)$.

Example 4. The example for Clay code have the parameters: $[n, k, d, \alpha, \beta, K] = [4, 2, 3, 4, 2, 8]$. After the encoding process the data is distributed across $n = 4$ nodes, from which $k = 2$ are data nodes and $n - k = 2$ are parity nodes. On each node is stored a superbyte consist of $\alpha = 4$ bytes. This makes the storage overhead $\frac{n\alpha}{k\alpha} = \frac{n}{k} = 2$ that is the ratio between the total number $n\alpha = 16$ of bytes stored to the number $K = k\alpha = 8$ of data bytes. When there is a failed node, the data downloaded from each of the $d = 3$ helper nodes are $\beta = 2$ bytes, which results in a normalized repair bandwidth of $\frac{d\beta}{k\alpha} = \frac{d}{k(d-k+1)} = 0.75$.

Data distribution process is done in several steps explained in [19]:

- A pair of coordinates to represent a layer are used
- Pairing of vertices and bytes
- Transforming from uncoupled to coupled-layer code
- Encoding the clay code
  - Load data into the 2 data nodes of coupled code
  - A pairwise reverse transformations used to be obtained the data stored in the 2 data nodes of uncoupled code
  - A MDS code is used in layer-by-layer fashion for determining the data stored in the parity nodes of uncoupled code
  - A pairwise forward transformation is used for obtaining the data to be stored in the parity nodes of coupled code

4 Analysis of the Different Comparison Procedures

This section gives analysis of the different comparison methods efficiency and validity. The four already described examples of codes Regenerating, Repetition, Reed-Solomon based DSS and Clay are considered in the comparison processes. The emphasis is put on the various chosen codes parameters and different file

sizes, this is done to be more evident the meaning of the usage of an adequate comparison method. The focus is put on the comparison technique that includes normalization of the total storage and total downloaded bandwidth with respect of the file size. The normalization operation enables comparison of codes with different file sizes and different parameters concerning the bandwidth and the storage.

Precisely, normalization is done over the file size defined as $|\mathcal{F}| = LK \log_2 q$, concerning the total bandwidth and the total storage given with,

$$T_\gamma = \beta dL \log_2 q, \tag{27}$$

$$T_\alpha = \alpha nL \log_2 q. \tag{28}$$

Thus, the axes on the graph to determine Normalized-Repair-bandwidth and Normalized-Total-storage, the Eqs. (27) and (28) need to be normalized over the file size and to become

$$N_\gamma = \frac{T_\gamma}{KL \log_2 q} = \frac{\beta dL \log_2 q}{KL \log_2 q} = \frac{\beta d}{K}, \tag{29}$$

$$N_\alpha = \frac{T_\alpha}{KL \log_2 q} = \frac{\alpha nL \log_2 q}{KL \log_2 q} = \frac{\alpha n}{K}. \tag{30}$$

4.1 Regeneration Code

Following *Example* 1 the regeneration code scheme has bandwidth of $\gamma = \beta d = 3$ symbols and storage-per-node of $\alpha = 2$ symbols, where $n = 4$ nodes and $k = 2$. Considering a file $\mathcal{F}$ of size $|\mathcal{F}|$ bytes (or $|\mathcal{F}|/\log_2 q = KL$ symbols), which is parsed into $L = 1$ blocks of $K = 4$ symbols then we get

$$N_\gamma^{Reg} = \frac{\beta d}{K} = \frac{3}{4} \text{ bytes}, \tag{31}$$

$$N_\alpha^{Reg} = \frac{\alpha n}{K} = 2 \text{ bytes}. \tag{32}$$

4.2 Repetition Code

Following *Example* 2 the repetition code has parameters defined as, an amount of downloaded data to repair a failed node $\gamma = \beta d = 3$, where $d = k = 1$ and $\beta = 3$ symbols and storage-per-node $\alpha = \beta = 3$ symbol, where $n = 3$. Considering a file of size $|\mathcal{F}| = LK \log_2 q$ bytes, where L is parsed into blocks of q-ary symbols and each block is with size of $K = 1$, we get

$$N_\gamma^{rep} = \frac{\beta d}{K} = 3 \text{ bytes}, \tag{33}$$

$$N_\alpha^{rep} = \frac{\alpha n}{K} = 9 \text{ bytes}. \tag{34}$$

4.3 RS Based DSS Code

By *Example* 3 the scheme bandwidth is given by $\gamma = \beta d = 3$, $k = 3$, $K = k = 3$, $d = k = 3$ and $n = 7$. The storage-per-node is $\alpha = 1$ symbol and the download bandwidth is $\beta = 1$.

With both measured in bytes, we get

$$N_{\gamma}^{RS} = \frac{\beta d}{K} = 1 \,\text{byte}, \tag{35}$$

$$N_{\alpha}^{RS} = \frac{\alpha n}{K} = \frac{7}{2} \,\text{bytes}. \tag{36}$$

4.4 Clay Code

Using *Example* 4 the bandwidth is $\gamma = \beta d = 6$, $k = 2$, $K = 8$, $d = 3$ and $n = 4$. The storage-per-node is $\alpha = 4$ symbol and $\beta = 2$.

With both measured in bytes, we get

$$N_{\gamma}^{Clay} = \frac{\beta d}{K} = \frac{3}{4} \,\text{bytes}, \tag{37}$$

$$N_{\alpha}^{Clay} = \frac{\alpha n}{K} = 2 \,\text{bytes}. \tag{38}$$

4.5 Comparison

In this section four codes Regeneration, RS based DSS, Repetition and Clay given in *Examples* 1, 2, 3 and 4 are compared. The set of values for each code are chosen to be different to show that the new proposed method does not depend on them to give real picture. Unlike the other, if we not care which parameters we compare, then the results might be unreliable. Three comparison methods Dimakis, Total and Normalized are applied on the same codes. All codes follows its unique code construction using various parameters, and most important all of them uses different file size $|\mathcal{F}|$. Dimakis comparison method compares the codes using only two parameters, α and γ, and the results from Fig. 4 shows that RS DSS is the best code and Clay worse. Total method adds two more values, n and L, providing a wider picture. According this method, RS DSS is the best, but the worst one now is the Repetition code. Normalized method normalize the values form the Total with respect of the code file size. With this approach now, the figure depicts that Clay and Regenerating code are the best. Analysing Fig. 4 we can see the difference among the various comparison methods and we can conclude that the Normalized method is the best choice for having an adequate result.

Figure 5 compares the same codes, just a little bit adjusted to have same file sizes. It is noticeable that all three comparison methods gives different outcomes.

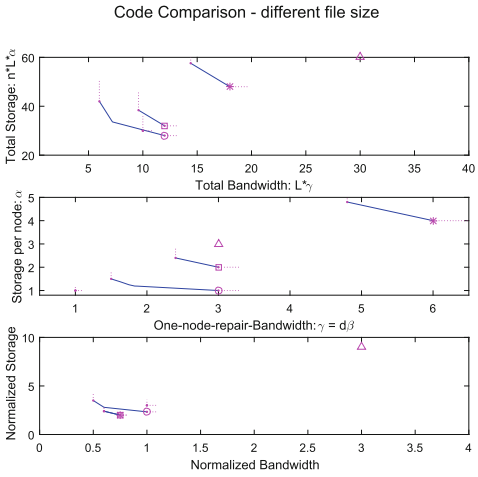

Fig. 4. Dimakis, Total and Normalized comparison methods for codes with different file size $|\mathcal{F}|$ bytes. Regeneration code (square) $(n = 4, k = 2, d = 3, \alpha = 2, \beta = 1, K = 4, L = 4)$, Repetition-DS-code (triangle) $(n = 3, k = 1, d = 1, \alpha = 3, \beta = 3, K = 1, L = 10)$, RS DSS-code (circle) $(n = 7, k = 3, d = 3, \alpha = 1, \beta = 1, K = 3, L = 4)$, and, Clay code (asterisk) $(n = 4, k = 2, d = 3, \alpha = 4, \beta = 2, K = 8, L = 3)$.

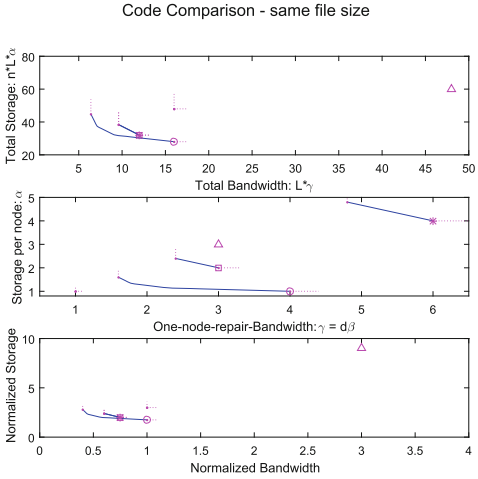

Fig. 5. Dimakis, Total and Normalized comparison methods for codes with same file size $|\mathcal{F}|$ bytes. Regeneration code (square) $(n = 4, k = 2, d = 3, \alpha = 2, \beta = 1, K = 4, L = 4)$, Repetition-DS-code (triangle) $(n = 3, k = 1, d = 1, \alpha = 3, \beta = 3, K = 1, L = 16)$, RS DSS-code (circle) $(n = 7, k = 4, d = 4, \alpha = 1, \beta = 1, K = 4, L = 4)$, and, Clay code (asterisk) $(n = 4, k = 2, d = 3, \alpha = 4, \beta = 2, K = 8, L = 2)$.

Dimakis says RS-DSS is best, and Clay worst code. Total method agrees with the best code, but in contrast says repetition is the one with poorest results. Normalization comparison claims that Regeneration and Clay code are better than the others, and Repetition is the worst.

Observing Figs. 4 and 5 we can see that changing the file size parameter in the codes affect the comparison methods Dimakis and Total by obtaining different results, unlike Normalized that doesn't change the final outcome.

5 Conclusion

A crucial challenge for distributed storage systems is the vast data generation, their processing and managing. Many researcher's work focuses to discover an efficient data management way to produce functional distributed storage networks. Various solutions in a form of data code constructions are proposed till now to deal this problem. Each newly developed code outperforms in some particular aspect of the DSS functioning, either it is the storage overhead, repair bandwidth, data distribution process or some other characteristic. Thus, in a general sense its very hard to say which code performs the best. Therefore, in this paper we are proposing a new comparison method, called Normalized, which allows more convenient approach for picking the best data code construction, despite the differently chosen codes parameters. The new approach produce more realistic results compare to the Dimakis and Total methods, while achieving good DSS efficiency. The comparison experiments are done using four different type of codes: Dimakis DSS, Repetition, Reed-Solomon based DSS and Clay code.

References

1. Moon, T.K.: Error Correction Coding: Mathematical Methods and Algorithms. Wiley-Interscience, Hoboken (2005)
2. MacWilliams, F.J., Sloane, N.J.A.: The Theory of Error-Correcting Codes. North Holland Mathematical Library, Amsterdam, The Netherlands (1983)
3. Rawat, A.S., Tamo, I., Guruswami, V., Efremenko, K.: MDS code constructions with small sub-packetization and near-optimal repair bandwidth. Trans. Inf. Theory IEEE **64**, 6506–6525 (2018)
4. Guruswami, V., Wootters, M.: Repairing Reed-Solomon codes. Trans. Inf. Theory IEEE **63**, 5684–5698 (2017)
5. Weatherspoon, H., Kubiatowicz, J.: Erasure coding vs. replication: a quantitative comparison. In: Proceedings of 1st International Workshop Peer-to-Peer System (IPTPS), pp. 328–338 (2001)
6. Sathiamoorthy, M., et al.: Xoring elephants: Novel erasure codes for big data. Proc. VLDB Endow. **6**, 325–336 (2013)
7. Memorandum of understanding for the implementation of the COST Action, European Cooperation for Statistics of Network Data Science (2015)
8. Dimakis*, A.G., Prabhakaran, V., Ramchandran, K.: Decentralized erasure code for distributed storage. Trans. Inf. Theory IEEE/ACM Netw., (2006)

9. Rawat, A.S., Koyluoglu, O.O., Silberstein, N., Vishwanath, S.: Optimal locally repairable and secure codes for distributed storage system. Info. Theory IEEE Trans. **60**, 212–236 (2013)
10. Rashmi, K.V., Shah, N.B., Kumar, P.V.: Regenerating codes for errors and erasures in distributed storage. In: Proceedings of IEEE International Symposium on Information Theory (ISIT) (2012)
11. Dimakis, A.G., Godfrey, P.B., Wu, Y., Wainright, M.J., Ramchandran, K.: Network coding for distributed storage systems. IEEE Trans. Inf. Theory **57**(8), 5227–5239 (2011)
12. Dimakis, A.G., Godfrey, P.B., Wainright, M., Ramchadran, K.: Network coding for distributed storage systems. In: Proceedings of 26th IEEE International Conference on Computer Communications, Anchorage, AK, pp. 2000–2008, May 2007
13. Han, Y.S., Zheng, R., Mow, W.H.: Exact regenerating codes for byzantine fault tolerance in distributed storage. In: INFOCOM Proceedings, pp. 2498–2506 (2012)
14. Han, Y.S., Pai, H.T., Zheng, R., Varshney, P.K.: Update-efficient regenerating codes with minimum per-node storage. In: Information Theory Proceedings (ISIT), pp. 1436–14406 (2013)
15. Goparaju, S., Tamo, I., Calderbank, R.: An improved sub-packetization bound for minimum storage regenerating codes. IEEE Information Theory Transactions, pp. 2770–2779 (2014)
16. Rashmi, K.V., Shah, N.B., Kumar, P.V.: Optimal exact-regenerating codes for distributed storage at the MSR and MBR points via a product-matrix construction. IEEE Trans. Inf. Theory **57**(8), 5227–5239 (2011)
17. Paunkoska, N., Finamore, W., Karamachoski, J., Puncheva, M., Marina, N.: Improving DSS Efficiency with Shortened MSR Codes. ICUMT (2016)
18. Paunkoska, N., Finamore, W., Marina, N.: Fair Comparison of DSS Codes. In: Future of Information and Communication Conference (FICC 2018) (2018)
19. Vajha, M., et al.: Clay codes: moulding MDS codes to yield an MSR code. In: Proceedings of the 16th USENIX Conference on File and Storage Technologies (FAST 2018), USENIX Association, USA, pp. 139–153 (2018)