

LDPC CODES FOR STORAGE SYSTEM

Field of the Invention

This invention relates to a system and a method to construct LDPC codes. More particular, this invention relates to a system and method of constructing LDPC codes and applying the LDPC codes to add redundancy for the storage system. Still more particularly, this invention relates to a system and method of constructing LDPC codes and applying the LDPC codes to newly added data to add redundancy for the storage system.

Prior Art

Storage system demands that the data is stored reliably so that lost data can be recovered even some of nodes or the storage medium are faulty, or become unavailable due to various reasons. This is realized by adding redundancy to the data stored. One type of technology is replication-based method realized by making multiple replicas. The replication based method suffers from inefficiency in terms of storage space. The other type of technology is the erasure code based method. For the erasure code based method, redundancy is added by additionally storing parity chunks generated from encoding of the original data. Various types of erasure codes may be used for generating redundant chunks. Some common types include Reed-Solomon code and low density parity check (LDPC) code. Reed-Solomon codes belong to the maximum distance separable (MDS) codes. Reed-Solomon codes minimize the storage space required for a certain level of reliability at the cost of high computational complexity and network traffic. Compared with Reed-Solomon codes, LDPC codes may achieve better trade-off between the storage efficiency, computational complexity and network traffic. However, various types of LDPC codes exist and conventional LDPC codes are not typically designed for

storage applications. These pose a few challenges when applying LDPC for the storage applications.

First, conventional erasure-code methods use LDPC codes with different sizes to
5 achieve trade-off between reliability and usages of system resources. For example, increasing the size of the erasure codes can provide higher reliability without compromising the spatial redundancy. However, the incurred complexities during encoding and decoding may cause longer latency and potentially heavy consumption of the precious network bandwidth. Conventional design of LDPC codes normally focuses
10 on an LDPC code with a particular size.

Secondly, erasure-code based methods pose further constraints in terms of how the divided chunks are placed compared with replication-based method. This is because, for an (n, m) erasure code which can tolerate e number of erasures where $e \leq m$, the n
15 chunks has to be put to at least n nodes to tolerate any e nodes' failure. By placing the chunks into more nodes may cause network traffic with longer distance. This leads to extensive consumption of precious network bandwidth such as cross-rack bandwidth and introduces longer delay for accessing the data as well.

20 Thirdly, the size of the data to be stored may be increased due to appending operations. One common scenario observed in the storage system is that a normal size files may be appended until it becomes very big. Unfortunately, the reliability of the whole file is dropped with the appending if a fixed n of the erasure codes is used. It is expected to have an increasing n after appending to keep up the reliability. However an LDPC code
25 is typically designed with a particular size. There is no efficient appending support for the erasure code based methods. In order to achieve that, the old encoded parity chunks have to be discarded and the appended file has to be re-encode. Re-encoding also leads

to potential over consumption of the system resources such as network bandwidth and Processing unit.

Hence, those skilled in the art are striving to provide an improved system and method that can construct tailored LDPC codes with variable sizes while still maintaining the encoding and decoding efficiencies, and to support efficient appending operation without re-encoding most of the encoded data.

Summary of the Invention

The above and other problems are solved and an advance in the art is made by a method and system to add redundancy in accordance with this invention. A first advantage of a method and system to add redundancy in accordance with this invention is that the method and system is easy to install and operate on current storage system. A second advantage of a method and system to add redundancy in accordance with this invention is that the method and system to add redundancy provides high reliability while maintaining the encoding and decoding complexities. A third advantage of a method and system to add redundancy in accordance with this invention is that the method and system to add redundancy arranges the encoded chunk in a certain manner which optimises the network bandwidth usage. A fourth advantage of a method and system to add redundancy in accordance with this invention is that the method and system to add redundancy provides efficient encoding of the data and higher reliability for data with increasing size.

In accordance with an aspect of the invention, an encoding method to add redundancy for data to be stored on a computer system or a distributed storage system is provided in the following manner. The method to add redundancy comprises receiving data to be encoded, selecting two integers n_b and m_b , where n_b/m_b is bigger than 1,

generating a parity check matrix, H , using the selected two integer satisfying the following conditions:

$$H(row, col) = \begin{cases} 0, & \text{when } 0 \ll idx < eta - 1, \text{ and} \\ & c \cdot (idx + 1) \ll col < n - (eta - idx - 1) \cdot c \\ 0, & \text{when } eta - 1 \ll idx < \frac{n-m}{c-b}, \text{ and} \\ & (0 \ll col < (idx - eta + 1) \cdot c \text{ or} \\ & (idx - eta + 1) \cdot c + n_b \ll col < n) \end{cases}$$

Where idx is integer, $row = idx \cdot (c - b), idx \cdot (c - b) + 1, \dots, idx \cdot (c - b) + c - b -$

$1, eta = \gcd(m_b, n_b), c = \frac{n_b}{eta}, b = \frac{m_b}{eta}$, and encoding the data using an erasure code with

5 the determined parity check matrix, H , to form encoded data and parity chunks.

In accordance with an embodiment of this invention, the step of generating a parity check matrix, H comprises selecting a set of parity check matrices $H_{b,t}$ as base codes, generating two sets of matrices $H_{u,t}$ and $H_{l,t}$ with the same dimensions $(n_b - m_b) \times$
 10 n_b , generating a set of parity check matrices, H_{rep} , with variable dimensions $(rep \times (n_b - m_b)) \times (rep \times n_b)$ based on the two sets of matrices $H_{u,t}$ and $H_{l,t}$.

In accordance with an embodiment of this invention, the step of selecting a set of parity check matrices $H_{b,t}$ as base codes with the dimensions $(n_b - m_b) \times n_b$ comprises
 15 generating a set of normal block codes with parity check matrices $H_{b,t}$ with the size $(n_b - m_b) \times n_b$, where $0 \leq t \leq \max_rep - 1$, and $\max_rep$ is an integer related to the maximum size of the generated parity check matrix, H .

In accordance with an embodiment of this invention, the step of generating two
 20 sets of matrices $H_{u,t}$ and $H_{l,t}$ with the same dimensions $(n_b - m_b) \times n_b$ comprises cutting each of the parity check matrices $H_{b,t}$ into two pieces with a cutting pattern such that $H_{b,t} = H_{u,t} + H_{l,t}$. Preferably, the cutting pattern is $eta = \gcd(m_b, n_b)$ and $b = m_b/eta, c = n_b/eta$, where c -units are moved to the right and down by $c-b$ unit.

In accordance with an embodiment of this invention, the encoding method further comprises assigning a node randomly to a first chunk of the encoded data and parity chunks, selecting a next chunk from the encoded data and parity chunks, determining a
5 subsequent node having the shortest network path with respect to the node, assigning the subsequent node having the shortest network path to the next chunk, repeating the step of selecting a next chunk from the encoded data and parity chunks for each subsequent chunk in the encoded data and parity chunks, determining a subsequent node having the shortest network path with respect to preceding node and assigning the
10 subsequent node having the shortest network path to the subsequent chunk, and transmitting the chunks to respective assigned nodes.

In accordance with an embodiment of this invention, the encoding method further comprises conducting column permutations by interlacing the columns associated with
15 the parity bits with those associated with the data bits with the following rule that inserting $(c-b)$ columns from $(m_b-1)^{\text{th}}$ columns of the base code $H_{b,t}$ in between every two b columns.

In accordance with an embodiment of this invention, the encoding method further
20 comprises receiving the size of the data to be encoded, FSize, and the chunk size of the encoded data, CSize, determining value of *rep1* using the FSize and CSize, determining the total size of the original data and the size of the new data to be appended to the original data, FSizeNew, determining value of *rep2* using the FSizeNew, determining a new parity check matrix H_{new} using *rep2*, re-encoding the first parity chunk and the
25 added chunks with the new parity check matrix H_{new} , and updating a meta data related to the appended data.

In accordance with an embodiment of this invention, the step of determining value of $rep1$ using $Fsize$ and $Csize$ comprises determining the smallest number, p , with the following expression, $p = Fsize/(Csize*m_b)$, setting $rep1 = p$, if p is an integer, and setting $rep1 =$ smallest integer that is bigger than p , if p is not an integer.

5

In accordance with an embodiment of this invention, the step of determining value of $rep2$ using the $FsizeNew$ comprises determining the smallest number, $pnew$, with the following expression, $pnew = FsizeNew/(Csize*m_b)$, setting $rep2 = pnew$, if $pnew$ is an integer, and setting $rep2 =$ smallest integer that is bigger than $pnew$, if $pnew$ is not an integer.

10

In accordance with an embodiment of this invention, the step of determining a new parity check matrix $Hnew$ using $rep2$ comprises generating two sets of matrices $H_{u,t}$ and $H_{l,t}$ with the same dimensions $(n_b - m_b) \times n_b$, generating a set of parity check matrices, H_{rep} , with variable dimensions $(rep2 \times (n_b - m_b) \times (rep2 \times n_b))$ based on the two sets of matrices $H_{u,t}$ and $H_{l,t}$, determining the new parity check matrix, $Hnew$, from the set of parity check matrices satisfying the following conditions:

15

$$H(row, col) = \begin{cases} 0, & \text{when } 0 \ll idx < eta - 1, \text{ and} \\ & c \cdot (idx + 1) \ll col < n - (eta - idx - 1) \cdot c \\ 0, & \text{when } eta - 1 \ll idx < \frac{n - m}{c - b}, \text{ and} \\ & (0 \ll col < (idx - eta + 1) \cdot c \text{ or} \\ & (idx - eta + 1) \cdot c + n_b \ll col < n) \end{cases}$$

Where idx is integer, $row = idx \cdot (c - b), idx \cdot (c - b) + 1, \dots, idx \cdot (c - b) + c - b -$

20 $1, eta = \gcd(m_b, n_b), c = \frac{n_b}{eta}, b = \frac{m_b}{eta}.$

In accordance with an embodiment of this invention, the method of encoding is implemented via an application program stored on a memory of a processing unit.

Brief Description of the Drawings

The above and other features and advantages in accordance with this invention are described in the following detailed description and are shown in the following drawings:

Figure 1 illustrating a representation of encoding a data chunk;

Figure 2 illustrating a representative of a recovering process;

Figure 3 illustrating a typical distributed storage system;

Figure 4 illustrating a processing system that executes the processes in accordance with an embodiment of this invention;

Figure 5 illustrating a flow diagram of a process for encoding a data in accordance with an embodiment of this invention;

Figure 6 illustrating a flow diagram of a process for assigning nodes to each of the chunks prior to storing on a distributed storage system in accordance with an embodiment of this invention;

Figure 7 illustrating a flow diagram of a process for encoding a data in accordance with an embodiment of this invention; and

Figure 8 illustrating a flow diagram of a process for re-encoding and appending encoded data chunks prior to storing on a distributed storage system in accordance with an embodiment of this invention.

Detailed Description

This invention relates to a system and a method to construct LDPC codes. More particular, this invention relates to a system and method of constructing LDPC codes and applying the LDPC codes to add redundancy for the storage system. Still more particularly, this invention relates to a system and method of constructing LDPC codes

and applying the LDPC codes to newly added data to add redundancy for the storage system.

It is envisioned that a system and a method in accordance with embodiments of this invention may be used for constructing LDPC codes for encoding data to be stored on a computer system or a distributed storage system. After the data is encoded with the LDPC codes constructed, the data chunks comprising parity and data chunks are arranged accordingly prior to distributing to the computer system or distributed storage system. Still further, for newly added data, such data together with at least one chunk of the encoded original data is encoded using a new parity check matrix in accordance with an embodiment of this invention. Such encoding of newly added data is more efficient and uses less resources of the storage while maintaining the integrity of the data.

LDPC code-based framework

Redundancy is required to provide reliability for storage systems such as storage arrays or distributed storage systems. For the erasure code based method, redundancy is added by additionally storing parity chunks generated from encoding of the original data. For an (n, m) erasure code, $n-m$ parity chunks are generated through encoding m chunks. Figure 1 illustrates data chunk 110 and $n-m$ parity chunk 120. As shown in Figure 1, $m=6$ and $n=9$. Each data chunks and parity chunks are stored in different nodes of the distributed storage system. In the event of a node failure, decoding process is engaged to gather all the required chunks to decode the lost chunk. Figure 2 illustrates the repair process for the same erasure code that is used in Figure 1. Assuming Node 4 storing data chunk D4 is down leading to the loss of D4. Another working node, Node 10 starts the repairing process by gathering all the chunks required to decode and recover the lost D4. In this illustration, Node 3, Node 5 and Node 7 are required to obtain D4. It is worth

mentioning for Reed Solomon code, all the data chunks are required even for single chunk lost. For LDPC codes, the number of the data chunks required is fewer.

In modern distributed storage system especially for big data storage, the storage nodes are arranged based on a certain network topology. Figure 3 illustrates a typical network architecture of a distributed storage system where multiple nodes are arranged on different racks.

The challenges of applying erasure codes method to the storage systems lie in the complexities brought by the required additional encoding and decoding processes. Various types of system resources are consumed for encoding and decoding. Two major types are computing resources and network bandwidth. It is often that the storage medium and processes are integrated within one platform. In other words, the computing resources and network bandwidths are shared among different types of jobs including storage related jobs and analysis related jobs. Thus, it is critical to minimize the consumption of the system resources for storage to allow more resources to be available for handling other jobs or maximize the overall system capacity. Compared with Reed-Solomon codes which use complex Galois Field operations, the encoding and decoding by LDPC code have the advantage of only involving with simple XOR operations. Hence, minimizing the number of XOR operations in encoding and decoding inevitably lead to reduction of consumption of the computing resources.

Network bandwidth is another important type of shared system resources. For the system with network topology shown in Figure 3, the types of network bandwidth are divided into two categories, namely, i) in-rack bandwidth (i.e. network bandwidth within the rack), and ii) cross-rack bandwidth (i.e. network bandwidth across the rack). Generally, cross-rack bandwidth is more precious than in-rack bandwidth as two nodes in the same

rack have more bandwidth and lower latency between each other than two nodes in two different racks. If the decoding process is taken as an example, different chunks of data are required to be gathered for decoding process, this gathering process incur the usage of network bandwidth and it is expected to reduce the total amount of traffic between
5 nodes, especially cross racks.

Figure 4 illustrates an exemplary processing system 400 that represents the processing systems in server, computer, and distributed storage system that execute instructions to perform the processes described below in accordance with this invention.

10 One skilled in the art will recognize that the instructions may be stored and/or performed as hardware, firmware, or software without departing from this invention. One skilled in the art will recognize that the exact configuration of each processing system may be different and the exact configuration executing processes in accordance with this invention may vary and processing system 400 shown in Figure 4 is provided by way of
15 example only.

Processing system 400 includes Central Processing Unit (CPU) 405. CPU 405 is a processor, microprocessor, or any combination of processors and microprocessors that execute instructions to perform the processes in accordance with the present invention.
20 CPU 405 connects to memory bus 410 and Input/Output (I/O) bus 415. Memory bus 410 connects CPU 405 to memories 420 and 425 to transmit data and instructions between the memories and CPU 405. I/O bus 415 connects CPU 405 to peripheral devices to transmit and receive data between CPU 405 and the peripheral devices. One skilled in the art will recognize that I/O bus 415 and memory bus 410 may be combined into one
25 bus or subdivided into many other busses and the exact configuration is left to those skilled in the art.

A non-volatile memory 420, such as a Read Only Memory (ROM), is connected to memory bus 410. Non-volatile memory 420 stores instructions and data needed to operate various sub-systems of processing system 400 and to boot the system at start-up. One skilled in the art will recognize that any number of types of memory may be used to perform this function.

A volatile memory 425, such as Random Access Memory (RAM), is also connected to memory bus 410. Volatile memory 425 stores the instructions and data needed by CPU 405 to perform software instructions for processes such as the processes for providing a system in accordance with this invention. One skilled in the art will recognize that any number of types of memory may be used to provide volatile memory and the exact type used is left as a design choice to those skilled in the art.

I/O device 430, keyboard 435, display 440, memory 445, network device 450 and any number of other peripheral devices connect to I/O bus 415 to exchange data with CPU 405 for use in applications being executed by CPU 405. I/O device 430 is any device that transmits and/or receives data from CPU 405. Keyboard 435 is a specific type of I/O that receives user input and transmits the input to CPU 405. Display 440 receives display data from CPU 405 and display images on a screen for a user to see. Memory 445 is a device that transmits and receives data to and from CPU 405 for storing data to a media. Alternatively, in the case of the storage server, memory 445 may be multiple storages that transmit and receive data to and from CPU 405 for storing data. Network device 450 connects CPU 405 to a network for transmission of data to and from other processing systems.

A system and method for encoding data received in accordance with an embodiment of the invention is described. Figures 5 and 6 illustrate the processes of encoding data and storing the encoded data on computing system or a distributed storage system. Figures 7 and 8 illustrate the processes of encoding the data and storing data on the distributed storage system.

Figure 5 illustrates a flow diagram of process 500 performed by the processor in processing system 400 in accordance with an embodiment of this invention. Process 500 is a process being executable upon receiving data for storing on a distributed storage system. Particularly, process 500 is a process for generating LDPC codes.

Process 500 begins with step 505 by selecting two integers n_b and m_b where $n_b/m_b > 1$ is required. The two integers n_b and m_b decide the redundancy rate. For example, if 1GB data is to be stored, at least $n_b/m_b > 1$ GB space is required. If 2GB data is to be stored, $2n_b/m_b$ has to be at least more than 2.

The parity check matrices generated should have zeroes at the following positions shown in the following Equation (1).

$$H(row, col) = \begin{cases} 0, & \text{when } 0 \ll idx < eta - 1, \text{ and} \\ & c \cdot (idx + 1) \ll col < n - (eta - idx - 1) \cdot c \\ 0, & \text{when } eta - 1 \ll idx < \frac{n-m}{c-b}, \text{ and} \\ & (0 \ll col < (idx - eta + 1) \cdot c \text{ or} \\ & (idx - eta + 1) \cdot c + n_b \ll col < n) \end{cases} \quad (1)$$

Where idx is integer, $eta = \gcd(m_b, n_b)$ and $b = m_b/eta$, $c = n_b/eta$, $row = idx \cdot (c - b), idx \cdot (c - b) + 1, \dots, idx \cdot (c - b) + c - b - 1$

It is worth mentioning that for purposes of this discussion, the indices of the matrix start from 0 instead of 1.

Hence, in step 510, process 500 generates at least a parity check matrix H satisfying the constraints shown in Equation (1) above. To assist this step of generating a parity check matrix H , the following steps may be performed:

1. Selecting a set of parity check matrices $H_{b,t}$ as base codes with the same dimensions $(n_b - m_b) \times n_b$

In particular, a set of normal block codes with parity check matrices $H_{b,t}$ with the size $(n_b - m_b) \times n_b$ are generated as the base codes, where $0 \leq t \leq \text{max_rep}-1$, where max_rep is an integer related to the maximum size of the generated parity check matrix, H . There are two important criteria to design the base codes. First, the minimum distance of the base code is expected to be as large as possible to guarantee a better performance for the derived LDPC Codes. Secondly, it is important to choose or design a parity check matrix $H_{b,t}$ with low row weightage as the row weightage is directly linked to repair complexity, as will be shown below.

2. Generating two sets of matrices $H_{u,t}$ and $H_{l,t}$ with the same dimensions $(n_b - m_b) \times n_b$

In particular, each of the parity check matrices $H_{b,t}$ is cut into two pieces. By filling out the other piece with zeros, two new sets of matrices are generated as $H_{u,t}$ and $H_{l,t}$, where $H_{b,t} = H_{u,t} + H_{l,t}$. Let $\text{eta} = \text{gcd}(m_b, n_b)$ and $b = m_b/\text{eta}$, $c = n_b/\text{eta}$. The cutting pattern is such that c -units are moved to the right and down by $c-b$ unit. An example is shown as follows with $m_b = 15$ and $n_b=21$.

Since $m_b = 15$ and $n_b=21$, $\text{eta} = \text{gcd}(15,21)=3$, $b=5$ and $c=7$. The line drawn on the parity check matrix H_b shows the cutting pattern.

$$H_b = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$

The H_u and H_l are as follows,

$$H_u = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$H_l = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

5

3. Generating a set of parity check matrices H_{rep} with variable dimensions (rep x $(n_b - m_b))$ x (rep x n_b)

After obtaining the two sets of matrices $H_{u,t}$ and $H_{l,t}$, a set of parity check matrices H_{rep} with variable dimensions (rep x $(n_b - m_b))$ x (rep x n_b) depending on the rep based on the two sets of matrices $H_{u,t}$ and $H_{l,t}$ is generated. In particular, H_{rep} are generated following the pattern shown in Equation (2). The rep is an integer and can be taken as the column or row or both column and row

10

combination. Some possible permutations of the variable dimensions ($\text{rep} \times (n_b - m_b)$) $\times$ ($\text{rep} \times n_b$) are as follows:

$$(\text{row} \times (n_b - m_b)) \times (\text{row} \times n_b)$$

$$(\text{col} \times (n_b - m_b)) \times (\text{col} \times n_b)$$

$$5 \quad (\text{row} \times (n_b - m_b)) \times (\text{col} \times n_b)$$

$$(\text{col} \times (n_b - m_b)) \times (\text{row} \times n_b)$$

where col is the column of the matrix and row is the row of the matrix.

10 It can be easily seen that the dimension of the generated H equals $(\text{rep} \times (n_b - m_b)) \times (\text{rep} \times n_b)$ and is decided by both the dimension of the base matrices and the repetition factor rep.

$$H = \begin{bmatrix} H_{l,0} & & & & H_{u,0} \\ H_{u,1} & H_{l,1} & & & \\ & H_{u,2} & H_{l,2} & & \\ & & \ddots & \ddots & \\ & & & H_{u,rep-1} & H_{l,rep-1}^{15} \end{bmatrix} \quad (2)$$

In another words, the parity check matrices generated should have zeroes at the following positions shown in the following Equation (1).

20 In step 515, process 500 encodes the data received using an erasure code with the parity check matrix H determined in step 510 to obtain the parity chunks and data chunks. The detailed processes of encoding data received using LDPC codes are generally known and would be omitted for brevity. Process 500 ends after step 515.

25 The advantages of applying process 500 to generate LDPC codes for storage applications will be discussed as follows.

1. Enabling effective design of LDPC codes with various sizes while maintaining the repair/decoding cost

The decoding complexity will be explained prior to the advantages of the generating of LDPC codes in accordance with process 500.

5

For each j^{th} bit, b_{ji} of the data chunk D_j , the respective bit of the parity chunks are generated through matrix multiplication. If the chunk size is assumed to be B_{size} , j is in the range of 0 to $B_{\text{size}} - 1$. b_{ji} represents the j^{th} bit of data chunk D_j . Since the same operation is done for each bit of the data chunk, j is omitted for simplicity for purposes of this discussion. The vector of encoded bits for n chunks is represented by $\vec{d} = [d_0, d_1, \dots, d_n]^T$, it is generated through $\vec{d}^T = \vec{b}G$ and satisfies $\vec{d}^T H = \vec{0}$.

10

The decoding is done by using the recovery equations determined based on the parity check matrix H . First, the decoding equation is determined and selected. Secondly, the node to store the repaired chunk determines the nodes storing the chunks required in the recovery equation to retrieve the respective chunks as illustrated by Figure 2. This is the step that brings the cluster high repair bandwidth leading to high consumption of network bandwidth. Thirdly, the XOR operations are conducted to compute the lost chunk. The major cost incurred during single repair cost is from the second and third steps. If an LDPC block code with parity check matrix shown in H_b above is taken as an example, the file has 21 chunks which are $\vec{d} = [d_0, d_1, \dots, d_{20}]^T$ and $\vec{d}^T H = \vec{0}$. If the node storing the first chunk d_0 is down, three recovery equations as listed below can be used to recover the first chunk d_0 .

15

20

25

$$d_0 = d_2 \oplus d_3 \oplus d_4 \oplus d_6 \oplus d_8 \oplus d_{14} \oplus d_{17} \quad (3)$$

$$d_0 = d_8 \oplus d_9 \oplus d_{12} \oplus d_{14} \oplus d_{15} \oplus d_{15} \oplus d_{16} \oplus d_{17} \oplus d_{18} \quad (4)$$

$$d_0 = d_1 \oplus d_3 \oplus d_7 \oplus d_{10} \oplus d_{16} \oplus d_{19} \quad (5)$$

Any one of the above equations from the three candidates can be used. Typically, the equation with the smallest number of elements is chosen. For the above example, equation (5) is preferred as the number of chunks and the number of XOR operations required is the smallest. The traffic occurred is the total bytes of all the chunks needed in the recovery equation. Based on the above example, at least 6 chunks are required for the repairing of the first chunk. The following equations (6)-(9) below illustrate the computational complexity and traffic cost of a single repair if the i^{th} chunk is lost. Equation (6) represents the number of XOR operations for recovering one chunk and it is closely related to the row weightages of H . For example, if equation (5) is used for repairing d_0 , the number of XOR operations is the i_0^{th} row weightages of H minus 2 which equates to 5. The chunks involved with the repair are those chunks with 1s in the rows where the i^{th} column is 1. As a result, the total traffic incurred is shown in Equation (9). For the same example shown in equation (5), repairing d_0 requires to transmit 6 other chunks including $d_1 d_3 d_7 d_{10} d_{16}$ and d_{19} . Equations (8) and (9) show the potential maximum number of XOR operations required and maximum traffic incurred during repair process.

$$XOR_DC_i = \left(\min_{i_0 \in U_i} \sum_{j=0}^{n-1} H[i_0, j] \right) - 2 \quad (6)$$

$$Traffic_DC_i = B_{size} \cdot (XOR_DC_i + 1) \quad (7)$$

$$\begin{aligned} XOR_DC_{max} = XOR_DC_i &= \left(\max_{1 \leq i \leq m} \sum_{j=0}^{n-1} H[i, j] \right) - 2 \\ &= \max(row_{weightage(H)}) - 2 \end{aligned} \quad (8)$$

$$Traffic_DC_{max} = B_{size} \cdot (XOR_DC_{max} + 1) \quad (9)$$

Thus, the row weightages of the H has to be minimized during the design to minimize the single repair cost when the n and m change even for the worst case. Therefore, for the generating of LDPC codes in process 500, it can be easily derived that

row weightages of the parity check matrix H are exactly the same as those of the base codes. With this analysis, the equation (8) above can be further simplified as

$$XOR_DC_{max} = \max(row_{weightage}(H)) - 2 = \max(row_{weightage}(H_{b,t})) - 2 \quad (10)$$

5

This means that as long as the row weightages of the base codes are designed carefully, higher reliability can be achieved by increasing repetition while maintaining the spatial efficiency and decoding complexity as low as those for the base codes. It is worth mentioning that a particular feature of the LDPC code generated by process 500 is that the row weightages of all the base codes are the same and the single repair complexity remains the same regardless of the value of n . It is also worth mentioning that all the base codes can be the same to simplify the process of selecting the base codes. In another word, only one base code needs to be selected.

10

15

2) Enabling effective usage of different types of network bandwidths

The total traffic incurred during the decoding process shown in equation (9) does not take the network topology into consideration. Based on the above decoding analysis, for repairing the i^{th} chunk, the chunks required are decided by the positions of the ones in the corresponding row where the i^{th} column is one for the parity check matrix. Since there exist the constraints of the proposed parity check matrix shown in the equations (1) and (2), it is also worth mentioning that the chunks potentially required for repairing i^{th} chunk can only come from the range of $(i \pm n_b - 1)^{th}$ chunk. If the chunks are placed based on the order indicated by the parity check matrix H , i.e. the neighbour chunks are put as close as possible. The chunks required for repairing process are all expected to be close to each other. Therefore, the encoded data is further processed by arranging the chunks according to the location of the nodes prior to distributing to the distributed storage system.

20

25

Figure 6 illustrates a flow diagram of process 600 performed by the processor in processing system 400 in accordance with an embodiment of this invention. Process 600 is a process being executable upon receiving encoded data from process 500. Particularly, process 600 is a process for re-ordering the chunks based on location of the nodes and distributing the chunk to various storages.

Process 600 begins with step 605 by receiving the parity and data chunks derived from process 500. In step 610, the first chunk is assigned a node randomly.

Process 600 then selects the next chunk from the parity and data chunks in step 615 and subsequently determines the shortest network path distance among the available nodes in respect to the previous assigned node in step 620. In step 625, the node with the shortest network path with respect to the previous assigned node is assigned to the chunk.

In step 630, process 600 determines whether there are any assigned chunks in the parity and data chunks. If all chunks are assigned a node, process 600 proceeds to step 635. Otherwise, process 600 repeats from step 615 to assign a node to the next chunk.

In step 635, the chunks are transmitted to distributed storage system based on the respective assigned nodes. Process 600 ends after step 635. It is worth mentioning that the chunks required for repairing the i^{th} chunk are independent from those required for $(i+2n_b)^{\text{th}}$ chunk. This property enables independent decoding to achieve better load balance and shorten the repairing latency.

For the distributed storage system, systematic codes are favoured over non-systematic codes because retrieving the original blocks does not require decoding for systematic codes. Hence, in another embodiment, another process to generate systematic version of LDPC code satisfying the above properties is introduced. For purposes of this discussion, the systematic version of the LDPC code is named as 'TAN Codes'. There are two more characteristics for the TAN Codes besides the properties illustrated in process 500 above. First, the TAN Codes are generated based on systematic base codes which can be expressed as follows.

$$H_b = [P^T | I_{n_b-m_b}]$$

Where P^T is a transpose of P matrix and P is any bit matrix containing elements either being 1 or 0 with the dimension of $(n_b-m_b) \times m_b$ and $I_{n_b-m_b}$ is the Identity matrix with the dimension of $(n_b-m_b) \times (n_b-m_b)$

Secondly, additional steps of column permutation have to be conducted before generating the two sets of matrices $H_{u,t}$ and $H_{l,t}$.

Figure 7 illustrates a flow diagram of process 700 performed by the processor in processing system 400 in accordance with an embodiment of this invention. Process 700 is a process being executable upon data to be stored on a distributed storage system. In particular, process 700 is a process illustrating the process of generating TAN Codes.

Process 700 begins with step 705 by selecting a set of parity check matrices as systematic base codes $H_{b,t}$ with the same dimensions $(n_b-m_b) \times n_b$.

In step 710, process 700 conducts column permutations. Particularly, a certain pattern of column permutation is proposed to be conducted on the systematic base codes $H_{b,t}$. For example, if a systematic LDPC block code with parity check matrix shown in H_b above is chosen as the base code with $m_b = 15$, $n_b = 21$. The column permutation is realized by interlacing the columns associated with the parity bits with those associated with the data bits. It follows the rules that inserting $(c-b)$ columns from $(m_b+1)^{\text{th}}$ columns of the original $H_{b,t}$ matrix in between every two b columns.

In step 715, two sets of matrices $H_{u,t}$ and $H_{l,t}$ with the same dimensions $(n_b - m_b) \times n_b$ are generated. In particular, each of the parity check matrices cut into two pieces. By filling out the other piece with zeros, two new sets of matrices are generated as $H_{u,t}$ and $H_{l,t}$, where $H_{b,t} = H_{u,t} + H_{l,t}$. Let $\text{eta} = \text{gcd}(m_b, n_b)$ and $b = m_b/\text{eta}$, $c = n_b/\text{eta}$. The cutting pattern is such that c -units are moved to the right and down by $c-b$ unit.

After obtaining the two sets of matrices $H_{u,t}$ and $H_{l,t}$, process 700 generates a set of parity check matrices H_{rep} with variable dimensions $(\text{rep} \times (n_b - m_b)) \times (\text{rep} \times n_b)$ depending on the rep based on the two sets of matrices $H_{u,t}$ and $H_{l,t}$ in step 720.

In step 725, process 700 determines at least a parity check matrix H satisfying the constraints shown in Equation (1) above. This is similar to step 510 in process 500.

In step 730, process 700 encodes the data received based on the parity check matrix H determined in step 725 to obtain the parity chunks and data chunks. The detailed processes of encoding data received using LDPC codes are generally known and would be omitted for brevity. This is similar to step 515 in process 500. Process 700 ends after step 730.

Additional advantages of applying the systematic TAN codes in process 700 for storage applications is that it enables effective design of systematic LDPC codes with various sizes while maintaining the encoding cost. To illustrate this advantage, the encoding complexity will first be explained prior to the advantages of the proposed systematic LDPC codes.

For each bit b_i of the data chunk D_i , the respective bits of the parity chunks are generated through matrix multiplication. The vector of encoded bits is represented by $\vec{d} = [d_0, d_1, \dots, d_m]^T$ and is generated through $\vec{d}^T = \vec{b}G$. For systematic codes, there is no need to encode the data block. Thus, $d_i = b_i$ where $0 \leq i \leq m$. Equation (11) below shows the encoding complexity for the i^{th} parity block, which is represented by the number of XOR operation needed,

$$XOR_En_i(G) = \left(\sum_{k=0}^{m-1} G[k, i + m] \right) \quad (11)$$

As all the parity chunks are required for encoding, the overall encoding complexity can be computed as the summation for all the required parity blocks as shown in Equation (12),

$$XOR_En_t(G) = \sum_{i=0}^{n-m-1} XOR_En_i(G) = \sum_{i=0}^{n-m} \sum_{k=0}^{m-1} G[k, i + m] \quad (12)$$

If it is assumed that encoding is done before distributing to the storage system. There is no additional traffic cost for encoding except for the added parity chunks. From the above analysis, it can be easily seen that the encoding complexity is largely dependent on the summation of the column weightage of the generator matrix, G . To be more specific, the encoding complexity is decided by the column weightage of sub-matrix P where P is a bit matrix containing elements being either 1 or 0. In brief, the relationship

between G , H and the complexities of the encoding and decoding can be summarised as follows.

	Row weightages of the H	→	The complexity of decoding
5	Column weightages of the G	→	The complexity of encoding

Therefore, if a family of LDPC codes is designed to maintain the column and row weightages of G and H , the complexities of repairing and encoding remain unchanged with the increase of n . For systematic codes, the row weightages of H and column weightages of G are only different by 1. Therefore, as long as the row weightages of H remain unchanged with the increase of n , both the complexity of encoding and decoding remains the same. In other words, the reliability can be increased with n without increasing the overall complexity in both decoding encoding.

15 Enabling efficient appending

One common scenario observed in the distributed storage system such as the Hadoop Distributed File System (HDFS) used in Hadoop, is that a normal size files may be appended until it becomes very big. However, if a fixed small block size is being used for appending the file, the reliability of the whole file is reduced. It can be observed that the parity chunks of the proposed TAN Codes are generated only based on the adjacent data chunks except for the first few parity chunks. Therefore, new H and G for the appended file can be easily derived with an increasing n by varying the repetition factor. The first few parity chunks are based on the last few data chunks as well. Therefore, only the first $(n_b - m_b) - (c - b)$ number of chunks is required to be re-encoded when rep is increased by any number, where $eta = gcd(m_b, n_b)$, $b = m_b/eta$ and $c = n_b/eta$. It can be observed that the number of chunks needed to be re-encoded is only dependent on the dimension of the base code. The percentage of the re-encoded parity chunks is (1-

1/eta)/rep. If eta=3 and rep=2 are taken as an example, 33% is required to be encoded instead of 100%. Thus, 67% improvement is achieved. As a result, if the repetition factors for the original data and the appended data are *rep1* and *rep2*, respectively, without applying the process 500 to obtain the LDPC codes, the total number of parity chunk to be encoded is $(n_b - m_b) \times rep2$. By using the process 500 to obtain the LDPC codes, there are only $(n_b - m_b) \times (rep2 - rep1 + (1 - 1/eta))$ chunks to be encoded. In another word, only $(rep2 - rep1 + (1 - 1/eta))/rep2\%$ of the encoding complexity is required by engaging the proposed TAN Codes. Therefore, TAN Codes supports efficient appending operation without re-encoding most of the encoded blocks. On the contrary, if a larger *n* of LDPC block code is applied for the appended file to avoid the drop of reliability, all the old parity chunks have to be re-encoded.

Figure 8 illustrates a flow diagram of process 800 performed by the processor in processing system 400 in accordance with an embodiment of this invention. Process 800 is a process being executable upon receiving encoded data from process 700. Particularly, process 800 is a process determining the size of the applied systematic TAN codes and encoding appended data with increasing block size.

Process 800 begins with step 805 by receiving the size of the data to be encoded, *Fsize*, and intended chunk size, *Csize*. The intended chunk size, *Csize*, is the encoded data chunk derived in process 700 while the size of the data to be encoded, *Fsize*, is the size of the original file.

In step 810, process 800 determines the smallest number, $p = Fsize / (Csize \times m_b)$. In step 815, if *p* is an integer, process 800 set *rep1*=*p* and proceeds to step 820. Otherwise, process 800 set *rep1* = smallest integer that is bigger than *p* and proceeds to step 820.

In step 820, process 800 determines the new increased size of the data to be encoded as $FSizeNew$. $FSizeNew$ is the size of the original data and the size of the new data to be appended to the original data.

5

In step 825, process 800 determines the smallest number, $pnew = FSizeNew/(Csize*m_b)$. In step 830, if $pnew$ is an integer, process 800 set $rep2 = pnew$ and proceeds to step 835. Otherwise, process 800 set $rep2 =$ smallest integer that is bigger than $pnew$ and proceeds to step 835.

10

In step 835, process 800 determines the new parity check matrix $Hnew$. The process of determining the new parity check matrix $Hnew$ is similar to steps 715-725 in process 700. In particular, the new parity check matrix $Hnew$ is determined in the following manner:

15

1. Two sets of matrices $H_{u,t}$ and $H_{l,t}$ with the same dimensions $(n_b - m_b) \times n_b$ are generated. In particular, each of the parity check matrices cut into two pieces. By filling out the other piece with zeros, two new sets of matrices are generated as $H_{u,t}$ and $H_{l,t}$, where $H_{b,t} = H_{u,t} + H_{l,t}$. Let $eta = gcd(m_b, n_b)$ and $b = m_b/eta$, $c = n_b/eta$. The cutting pattern is such that c -units are moved to the right and down by $c-b$ unit.
2. After obtaining the two sets of matrices $H_{u,t}$ and $H_{l,t}$, a set of parity check matrices H_{rep} with variable dimensions $(rep2 \times (n_b - m_b)) \times (rep2 \times n_b)$ based on the two sets of matrices $H_{u,t}$ and $H_{l,t}$ is generated.
3. The new parity check matrix $Hnew$ is one that satisfies the constraints shown in Equation (1) above.

20

25

In step 840, process 800 re-encodes the first $(n_b - m_b) - (c - b)$ parity chunks and the newly added chunks with the new parity check matrix H_{new} .

In step 845, the meta data related to the appended file is updated accordingly.

5

The above is a description of exemplary embodiments of a system and method in accordance with this invention. It is foreseeable that those skilled in the art can and will design alternative systems based on this disclosure that infringe upon this invention as set forth in the following claims.

10

Claims:

1. An encoding method for adding redundancy to data to be stored on a computer system or a distributed storage system comprising:

receiving data to be encoded;

5 selecting two integers n_b and m_b , where n_b/m_b is more than 1;

generating a parity check matrix, H , using the selected n_b and m_b satisfying the following conditions:

$$H(row, col) = \begin{cases} 0, & \text{when } 0 \ll idx < eta - 1, \text{ and} \\ & c \cdot (idx + 1) \ll col < n - (eta - idx - 1) \cdot c \\ 0, & \text{when } eta - 1 \ll idx < \frac{n-m}{c-b}, \text{ and} \\ & (0 \ll col < (idx - eta + 1) \cdot c \text{ or} \\ & (idx - eta + 1) \cdot c + n_b \ll col < n) \end{cases}$$

Where idx is integer, $row = idx \cdot (c - b), idx \cdot (c - b) + 1, \dots, idx \cdot (c - b) + c - b -$

1, $eta = \gcd(m_b, n_b), c = \frac{n_b}{eta}, b = \frac{m_b}{eta}$, and

10 encoding said data using an erasure codes with the said determined parity check matrix, H , to form encoded data and parity chunks.

2. The method according to claim 1, wherein said generating a parity check matrix, H , comprises:

15 selecting a set of parity check matrices $H_{b,t}$ as base codes with the dimensions $(n_b - m_b) \times n_b$

generating two sets of matrices $H_{u,t}$ and $H_{l,t}$ with the same dimensions $(n_b - m_b) \times n_b$; and

20 generating a set of parity check matrices, H_{rep} , with variable dimensions $(rep \times (n_b - m_b)) \times (rep \times n_b)$ based on the two sets of matrices $H_{u,t}$ and $H_{l,t}$.

3. The method according to claim 2, wherein said selecting a set of parity check matrices $H_{b,t}$ as base codes with the dimensions $(n_b - m_b) \times n_b$ comprises:

generating a set of normal block codes with parity check matrices $H_{b,t}$ with the size $(n_b - m_b) \times n_b$, where $0 \leq t \leq \text{max_rep}-1$, and max_rep is an integer related to the maximum size of the generated parity check matrix, H .

- 5 4. The method according to claim 2, wherein said generating two sets of matrices $H_{u,t}$ and $H_{l,t}$ with the same dimensions $(n_b - m_b) \times n_b$ comprises:

cutting each of said parity check matrices $H_{b,t}$ into two pieces with a cutting pattern such that $H_{b,t} = H_{u,t} + H_{l,t}$.

- 10 5. The method according to claim 4 wherein said cutting pattern is $\text{eta} = \text{gcd}(m_b, n_b)$ and $b = m_b/\text{eta}$, $c = n_b/\text{eta}$, where c-units are moved to the right and down by c-b unit.

6. The method according to claim 1 further comprising:

assigning a node randomly to a first chunk of said encoded data and parity chunks;

- 15 selecting a next chunk from said encoded data and parity chunks;

determining a subsequent node having the shortest network path with respect to said node;

assigning said subsequent node having the shortest network path to said next chunk;

- 20 repeating said step of selecting a next chunk from said encoded data and parity chunks for each subsequent chunk in said encoded data and parity chunks, determining a subsequent node having the shortest network path with respect to preceding node and assigning said subsequent node having the shortest network path to said subsequent chunk; and

- 25 transmitting said chunks to respective assigned nodes.

7. The method according to claim 2 further comprising:

conducting column permutations.

8. The method according to claim 7 wherein said conducting column permutations comprises:

5 interlacing the columns associated with the parity bits with those associated with the data bits with the following rule that inserting $(c-b)$ columns from $(m_b-1)^{\text{th}}$ columns of the base code $H_{b,t}$ in between every two b columns.

9. The method according to claim 8 further comprises:

10 receiving the size of the data to be encoded, FSize, and the chunk size of the encoded data, CSize;

determining value of $rep1$ using said FSize and CSize;

determining the total size of the original data and the size of the new data to be appended to the original data, FSizeNew;

15 determining value of $rep2$ using said FSizeNew;

determining a new parity check matrix H_{new} using $rep2$;

re-encoding the first $(n_b-m_b)-(c-b)$ parity chunks and the added chunks with the new parity check matrix H_{new} ; and

updating a meta data related to the appended data.

20

10. The method according to claim 9 wherein said step of determining value of $rep1$ using Fsize and Csize comprises:

determining the number, p , with the following expression

$$p = \text{FSize}/(\text{CSize} * m_b);$$

25 setting $rep1 = p$, if p is an integer; and

setting $rep1$ equals to the smallest integer that is bigger than p , if p is not an integer.

11. The method according to claim 10 wherein said step of determining value of $rep2$ using said $FSizeNew$ comprises:

determining the number, $pnew$, with the following expression

$$pnew = FSizeNew / (CSize * m_b);$$

setting $rep2 = pnew$, if $pnew$ is an integer; and

setting $rep2$ equals to the smallest integer that is bigger than $pnew$, if $pnew$ is not an integer.

12. The method according to claim 11 wherein said determining a new parity check matrix $Hnew$ using $rep2$ comprises:

generating two sets of matrices $H_{u,t}$ and $H_{l,t}$ with the same dimensions $(n_b - m_b) \times n_b$;

generating a set of parity check matrices, H_{rep} , with variable dimensions $(rep2 \times (n_b - m_b) \times (rep2 \times n_b))$ based on the two sets of matrices $H_{u,t}$ and $H_{l,t}$;

determining said new parity check matrix, $Hnew$, from said set of parity check matrices satisfying the following conditions:

$$H(row, col) = \begin{cases} 0, & \text{when } 0 \ll idx < eta - 1, \text{ and} \\ & c \cdot (idx + 1) \ll col < n - (eta - idx - 1) \cdot c \\ 0, & \text{when } eta - 1 \ll idx < \frac{n - m}{c - b}, \text{ and} \\ & (0 \ll col < (idx - eta + 1) \cdot c \text{ or} \\ & (idx - eta + 1) \cdot c + n_b \ll col < n) \end{cases}$$

Where idx is integer, $row = idx \cdot (c - b), idx \cdot (c - b) + 1, \dots, idx \cdot (c - b) + c - b -$

$$1, eta = \gcd(m_b, n_b), c = \frac{n_b}{eta}, b = \frac{m_b}{eta}.$$

13. A system for encoding a data and adding redundancy to data to be stored on a computer system or a distributed storage system comprising:

a processing unit having a processor and memory;

instructions stored on said memory executable by said processor to:

receiving data to be encoded;

selecting two integers n_b and m_b , where n_b/m_b is more than 1;

5 generating a parity check matrix, H , using said selected two integers n_b and m_b satisfying the following conditions:

$$H(row, col) = \begin{cases} 0, & \text{when } 0 \ll idx < eta - 1, \text{ and} \\ & c \cdot (idx + 1) \ll col < n - (eta - idx - 1) \cdot c \\ 0, & \text{when } eta - 1 \ll idx < \frac{n-m}{c-b}, \text{ and} \\ & (0 \ll col < (idx - eta + 1) \cdot c \text{ or} \\ & (idx - eta + 1) \cdot c + n_b \ll col < n) \end{cases}$$

Where idx is integer, $row = idx \cdot (c - b), idx \cdot (c - b) + 1, \dots, idx \cdot (c - b) + c - b - 1$, $eta = \gcd(m_b, n_b)$, $c = \frac{n_b}{eta}$, $b = \frac{m_b}{eta}$, and

encoding said data using an erasure code with the said determined parity

10 check matrix, H , to form encoded data and parity chunks.

14. The system according to claim 13, wherein said instruction to generating a parity check matrix, H , comprises instructions to:

selecting a set of parity check matrices $H_{b,t}$ as base codes with the dimensions

15 $(n_b - m_b) \times n_b$

generating two sets of matrices $H_{u,t}$ and $H_{l,t}$ with the same dimensions $(n_b - m_b) \times$

n_b ; and

generating a set of parity check matrices, H_{rep} , with variable dimensions $(rep \times (n_b - m_b) \times (rep \times n_b))$ based on the two sets of matrices $H_{u,t}$ and $H_{l,t}$.

20

15. The system according to claim 14, wherein said instruction to selecting a set of parity check matrices $H_{b,t}$ as base codes with the dimensions $(n_b - m_b) \times n_b$ comprises instruction to generating a set of normal block codes with parity check matrices $H_{b,t}$

with the size $(n_b - m_b) \times n_b$, where $0 \leq t \leq \text{max_rep}-1$, and max_rep is an integer related to the maximum size of the generated parity check matrix, H .

16. The system according to claim 14, wherein said instruction to generating two sets of matrices $H_{u,t}$ and $H_{l,t}$ with the same dimensions $(n_b - m_b) \times n_b$ comprises instruction to cutting each of said parity check matrices $H_{b,t}$ into two pieces with a cutting pattern such that $H_{b,t} = H_{u,t} + H_{l,t}$.

17. The system according to claim 16 wherein said cutting pattern is $\text{eta} = \text{gcd}(m_b, n_b)$ and $b = m_b/\text{eta}$, $c = n_b/\text{eta}$, where c -units are moved to the right and down by $c-b$ unit.

18. The system according to claim 13 further comprising:

instructions stored on said memory executable by said processor to:

assigning a node randomly to a first chunk of said encoded data and parity chunks;

selecting a next chunk from said encoded data and parity chunks;

determining a subsequent node having the shortest network path with respect to said node;

assigning said subsequent node having the shortest network path to said next chunk;

repeating said step of selecting a next chunk from said encoded data and parity chunks for each subsequent chunk in said encoded data and parity chunks,

determining a subsequent node having the shortest network path with respect to preceding node and assigning said subsequent node having the shortest network path to said subsequent chunk; and

transmitting said chunks to respective assigned nodes.

19. The system according to claim 14 further comprising:

instructions stored on said memory executable by said processor to:

conducting column permutations.

5 20. The system according to claim 19 wherein said instruction to conducting column permutations comprises instruction to interlacing the columns associated with the parity bits with those associated with the data bits with the following rule that inserting $(c-b)$ columns from $(m_b-1)^{\text{th}}$ columns of the base code $H_{b,t}$ in between every two b columns.

10

21. The system according to claim 20 further comprises:

instructions stored on said memory executable by said processor to:

receiving the size of the data to be encoded, FSize, and the chunk size of the encoded data, CSize;

15

determining value of $rep1$ using said FSize and CSize;

determining the total size of the original data and the size of the new data to be appended to the original data, FSizeNew;

determining value of $rep2$ using said FSizeNew;

determining a new parity check matrix H_{new} using $rep2$;

20

re-encoding the first $(n_b-m_b)-(c-b)$ parity chunks and the added chunks with the new parity check matrix H_{new} ; and

updating a meta data related to the appended data.

22. The system according to claim 21 wherein said instruction to determining value of

25

$rep1$ using Fsize and Csize comprises instructions to:

determining the number, p , with the following expression

$$p = \text{FSize}/(\text{CSize} * m_b);$$

setting $rep1 = p$, if p is an integer; and

setting $rep1$ equals to the smallest integer that is bigger than p , if p is not an integer.

5 23. The system according to claim 22 wherein said instruction to determining value of $rep2$ using said $FSizeNew$ comprises instructions to:

determining the number, $pnew$, with the following expression

$$pnew = FSizeNew / (CSize * m_b);$$

setting $rep2 = pnew$, if $pnew$ is an integer; and

10 setting $rep2$ equals to the smallest integer that is bigger than $pnew$, if $pnew$ is not an integer.

24. The system according to claim 23 wherein said instruction to determining a new parity check matrix $Hnew$ using $rep2$ comprises instructions to:

15 generating two sets of matrices $H_{u,t}$ and $H_{l,t}$ with the same dimensions $(n_b - m_b) \times n_b$;

generating a set of parity check matrices, H_{rep} , with variable dimensions $(rep2 \times (n_b - m_b) \times (rep2 \times n_b))$ based on the two sets of matrices $H_{u,t}$ and $H_{l,t}$;

20 determining said new parity check matrix, $Hnew$, from said set of parity check matrices satisfying the following conditions:

$$H(row, col) = \begin{cases} 0, & \text{when } 0 \ll idx < eta - 1, \text{ and} \\ & c \cdot (idx + 1) \ll col < n - (eta - idx - 1) \cdot c \\ 0, & \text{when } eta - 1 \ll idx < \frac{n - m}{c - b}, \text{ and} \\ & (0 \ll col < (idx - eta + 1) \cdot c \text{ or} \\ & (idx - eta + 1) \cdot c + n_b \ll col < n) \end{cases}$$

Where idx is an integer, $row = idx \cdot (c - b), idx \cdot (c - b) + 1, \dots, idx \cdot (c - b) + c - b - 1$, $eta = \gcd(m_b, n_b)$, $c = \frac{n_b}{eta}$, $b = \frac{m_b}{eta}$.