A Fast Map-Reduce Algorithm for Burst Errors in Big Data Cloud Storage

Xue Qin
Department of ECE
University of Texas at San Antonio
San Antonio, TX, U.S.A.
qinxue1107@gmail.com

Brian Kelley
Department of ECE
University of Texas at San Antonio
San Antonio, TX, U.S.A.
Brian.Kelley@utsa.edu

Mahdy Saedy
AT&T-Services Design & Dev.
AT&T Research Labs
Middletown, NJ 07748 USA
Mahdy.Saedy@att.com

Abstract - *In distributed storage for Big Data systems, there is a need for exact repair, high bandwidth codes. The challenge for exact repair in big-data storage is to simultaneously enable both very high bandwidth repair using Map-Reduce and simple coding schemes that also combine robust maximally distance separable (MDS) exact repair. MDS repair is for the rare, but exceptional outlier error patterns requiring optimum erasure code reconstruction. We construct the optimum fast bandwidth repair for big-data sources. Our system uses Map-Reduce, exact repair reconstruction. The algorithm combines MDS with a second fast decode algorithm in a cloud environment. We illustrate cloud experiments for optimum fast bandwidth reconstruction for 1-Exabyte Big Data in the cloud and demonstrate cloud results for Poisson error rate arrival models. Unlike prior methods, we jointly solve the problem of fast bandwidth repair for burst-memory error patterns and for code rates up to $\frac{2}{3}$ in a real time error model framework for Big Data. Furthermore, simulations indicate this method outperforms prior fast bandwidth approaches for burst errors. We also illustrate Map-Reduce algorithm optimized for fast bandwidth repair in Big Data storage in clouds.*

Keywords: Exabyte Big Data, fast reconstruction, exact repair, erasure codes, Map-Reduce, cloud systems, maximally distance separable

1 Introduction

Cloud techniques have relevance to many engineering research fields. One of the most important applications for using the cloud is "Big Data." Since 2012, every day 2.5 Exabytes (2.5×10^{18} bytes) of data are created. Such tremendous volumes of "Big Data" are increasingly common. Industrial design of data centers to process Big Data and the use of cloud servers for distributed implementation often invoke open-source applications such as Hadoop. Hadoop is one of the most useful open-source software frameworks for distributed storage and processing [1]-[5] of Big Data, particularly on clusters of commodity hardware. In Hadoop Distributed File System (HDFS), files are split into large blocks (default 64MB or 128MB) and are distributed into the blocks amongst the Data Nodes in the cluster. In case of data loss, HDFS replicates the original data for three times to provide a robust storage.

Distributed storage systems in cloud data centers, Data as a Service (DaaS) systems, and other Big Data systems increasingly require fast bandwidth node reconstruction in the event of disk failures. For instance, open-source frameworks such as OpenStack [6] deploy Swift DaaS and Glance image storage. In many instances, there is a need to both achieve optimum guarantees of data recovery, in the event of node failure *and* fast bandwidth repair. Degree of protection and speed *are countervailing objectives*.

1.1 System Model for Big Data Reconstruction

Typically, an erasure code, converts k information symbols using a generator matrix into an N-symbol codeword [7],[8]. We propose the use of maximum distance separable (MDS) coding [9],[10] jointly with a fast reconstruction algorithms for correction, *but reformulated in a cloud environment for Big-Data* [11]. The specific pattern of the errors in memory determines whether we can apply fast bandwidth or the optional MDS reconstruction.

Given a Poisson error arrival patterns into the Big Data system, a core concept is this—control the mean number of errors injected into the Big Data set by applying fast erasure decoding in real time in the cloud at a sufficient rate, $1/\mathrm{T}_{per}$ so that slower MDS reconstruction occurs with low probabilistic guarantees. We define the effective bandwidth achieving this objective, $\mathrm{EffBW} = 1/\mathrm{T}_{per}$. We apply the fast algorithm with probability,$q = f(T_{per}, \lambda)$, thereby enabling mean computational latency of the reconstruction to become

$$\mathrm{D}_{\mu} = (1 - \mathrm{q}) \times \mathrm{L} \times \mathrm{T} + \mathrm{q} \times \mathrm{T} \ll \mathrm{L} \times \mathrm{T} \qquad (1)$$

In Table 1, T and $\mathrm{L} \times \mathrm{T}$ are the computational latency of the fast bandwidth reconstruction and (L-times) longer MDS latency, respectively. A key optimization goal is the determination of the mean upper bound time T_{per} so that the combined rate $\frac{2}{3}$ simple regenerative code with MDS [12] in a cloud environment is dominated by the simple regenerative code latency (e.g. $\mathrm{D}_{\mu}, \ll \mathrm{T}_{per}$). We define our optimality objective as the upper bound correction period, T_{per}, enabling the mean computational delay of the reconstruction system, D_{μ}, to be 1% of T_{per}. Figure 1 illustrates that the fast algorithm delay. This occurs with high probabilistic guarantees, greatly decreasing $E[D_\mu]$.

The system model can be used to provide the mean computational delay of the overall reconstruction system,D_μ, to be 1% of T_per. This is based upon a given error arrival rate, λ_0, in memory and the dataset size A. That is, given a known λ_0, and A, the system model will provides the optimal options for (N,M) pair to keep the $D_\mu \approx 1\% \times$T_per. Constructing the Big Data structure by using these system model parameters reduces the time cost for exact repair of lost data (nodes). For example, let us assume the system node size is equal to the Hadoop node size. Since Hadoop repairs the lost nodes in parallel and our system is also running our Hadoop fast bandwidth algorithm, the time cost for single node and multiple node repairs are the same. However, since we do not replicate the original data by a factor of three, our system framework significantly reduces the memory storage.

Section 2 proposes a new Map-Reduce, fast-bandwidth regenerative algorithm for Big Data on Clouds. It is based upon a Poisson error arrival model that we have simulated. Section 3 describes fast bandwidth system architecture four our distributed, fast bandwidth regenerative code capable of large error burst handling. Section 3 also outlines the variety of cloud based processing protocols to be implemented as applications. A new algorithm that defines the minimum set of MDS corrections to enable fast bandwidth via simple regeneration is presented using a Hadoop process. Section 4 summarized our conclusions.

2 Fast Bandwidth Cloud Regeneration of Big Data

Table 1 represents the set of parameters used in our cloud framework. Let's assume that our Big Data set over time has a Poisson error arrival rate in memory of λ $\text{bit errors/sec/512 bytes}$ [13]. Let's define the optimum effective bandwidth of cloud reconstruction of the Big Data set,$\mathcal{A}$, as the value T_{per} that enables D_μ to be 1% of T_{per}

STEP 1: For a given λ and Big Data size, $\mathcal{A}$, we determine the optimum Map-Reduce parallelism P. Based upon λ and the size of $\mathcal{A}$, divide the set $\mathcal{A}$ into $\mathcal{A}$ /P memories.

STEP 2: Encode via MAP Reduce each of the $\mathcal{A}$/P memories with a combined rate $\frac{2}{3}$ simple regenerative code and MDS code. The MDS code further divides each of the $\mathcal{A}$/P memories into N nodes based upon the (N,k) encoding for MDS.

1. For the fast bandwidth sparse encode procedure, form N encode data sets containing (x,y,s) triplets (see [8]).
2. The (N,k) MDS encoding procedure divides a cloud database of size $\mathcal{A}/P$ into k sets of size $\frac{\mathcal{A}}{(P\times k)}$. In each system, the resulting source data is further encoded into N distributed memories via an (N,k) code with an erasure code rate, $r=\frac{k}{N}$. We therefore allocate $P\times N$ memories of size $\frac{\mathcal{A}}{(P\times N)}$ in a Map-Reduce framework. From this construction, we can tolerate $N-k$ disk failures in each of the P sets and still reconstruct any of the k information nodes in each set.

STEP 3: As Poisson errors arrive at the Big Data memory, For $\text{j} = 1,2,\ldots \text{P}$

3. Apply reconstruction, adaptively selecting fast bandwidth with a probablity q or, equivalently, MDS with a probablity $1-\text{q}$. The mean computational latency of the reconstruction code is therefore $1\% \times T_{per}$ ($\ll$ latency of the MDS).
4. Place the $\text{N}\ (\text{x},\text{y},\text{s})$ encoded data sets in N separate Nodes for each j.

EndFor

We illustrate this cloud application architecture in Figure 2 and the cloud system protocol in Figure 3.

2.1 Hadoop Big Data Reconstruction on Clouds

Our goal is fast real time processing of Big-Data with reconstruction speed increased using Hadoop parallel operation and Fast Bandwidth repair. The Map-reduce pseudo code is defined as follows:

Map pseudocode:

```
Map
    Define Map(Node_input N, detection_input D):
        Correcting Set = [];
            for each Node in Node_input N
            if detection flag D(i) == 1, then
          CorrectingSet = [Node((i-2)%N);
                    Node((i-1)%N); Node(i);
                  Node((i+1)%N);
Node((i+2)%N)];
Error_Location = i;
            endif
              endfor
    return Reduce (Error_Location, CorrectingSet);
```

Reduce pseudocode

```
Reduce
   Define  Reduce(Error_Location key,  CorrectingSer
value):
        New_node = [];
        New_node.x = xor(Node((i-2)%N).s , Node((i-
        2)%N).y);
        New_node.y = xor(Node((i+1)%N).x,  Node((i-
        1)%N).s);
        New_node.s = xor(Node((i+2)%N).x ,
        Node((i+1)%N).y);
        Node(i) = New_node;
   return (Error_Location, New_node)
```

The Cloud Simulation Results in Fast Bandwidth Reconstruction, as illustrated in Figure 4. In a Hadoop cluster (Hadoop 1.0 configuration), there are *Name Nodes* and *Data Nodes.* The data replication on each data node is 64MB. The MDS encoding procedure divides a database of size $\mathcal{A}$ into k sets of size $\frac{\mathcal{A}}{k}$. This configuration needs to be replicated to all data nodes blocks (see Fig. 3). The resulting data is mapped to n distributed memories via an (n, k) code of rate $r = \frac{k}{n}$. From this construction, we can tolerate $n - k$ disk failures, or erasures, and still reconstruct any of the k information nodes. A second key objective is fast reconstruction. MDS decoding is not necessarily supportive of fast, real time data reconstruction. One method proposed for fast reconstruction has been the joint combining of a rate $\frac{2}{3}$ simple regenerative codes with MDS. Under this revised model n storage nodes can tolerate $n - k$ erasures at a rate $\frac{2}{3} \times \frac{k}{n}$ and arbitrarily capable of achieving rates up to $r = \frac{2}{3}$. Besides the Simple Regenerative Code (SRC), there are many other codes based on MDS Codes or Reed-Solomon Code that support distributed storage. Different open-source erasure coding libraries provide different performance.

2.2 Fast Bandwidth Simple Regenerative Code

The prior simple regenerative method does not handle burst errors. A principle innovation described here is the development of a simple regenerative code capable of correcting burst errors in a fast way. The method is orders of magnitude faster than the prior approach, due to our ability to avoid slower MDS recovery for all but the most severe scenarios. We rely on the use of a simple regenerative extension field coupled to MDS, capable of fast burst reconstructions, simultaneously if needed, with MDS.

3 Derivation of Protocols for MDS Pre-coding and Sparse Codes

Fig. 5 shows the construction of our storage nodes from information source. The source is pre-coded using Maximum Distance Separable (MDS) method. Then additional redundancy in node is generated by a subsequent simple regenerative Sparse Code defined in in $\mathrm{GF}(2^p)$. Figure 6 displays the details of how MDS pre-coding and Sparse Code working. In the pre-coding part, information source is separated into two parts which have the same length and then go through the (n, k) MDS encoding. After this encoding, two encoded information sets (x, y) are generated. Then, for each pair of (x, y), we apply an finite field addition operation in $\mathrm{GF}(2^p)$. The relationship between x, y and f is shown in Eq. 1

$$x = Gf^{(0)} \tag{1}$$

$$y = Gf^{(1)} \tag{2}$$

Here, G is the generator matrix of the (n, k) MDS code in GF(2^p). The parameters x and y are the encoded vector with length $n \times p$. Inputs $f^{(0)}$ and $f^{(1)}$ are two independent information sources with length $k \times p$. Output s is defined by

$$s = Gf^{(0)} + Gf^{(1)} = G\left(f^{(0)} + f^{(1)}\right) = x + y$$

In GF(2) addition in $s = x \oplus y$ implies a XOR. But we define addition, $\boxplus$, in the extension field of $\mathrm{GF}(2^p)$. For example in $\mathrm{GF}(2^5)$, with primitive polynomial $p(X) = 1 + X^2 + X^5$, $x_0 = \alpha^5 = [10100]$, $y_0 = \alpha^{11} = [11100]$. Then $x_0 \boxplus y_0 = [01000] = \alpha$ in $\mathrm{GF}(2^5)$ addition under $p(X)$. This is shown in Fig. 3.

3.1 Simple Regenerative Code for FAST Bandwidth Construction of Burst Errors

After encoding the information and redundant data, we place order the data in a defined sequence so that neighboring nodes enable fast bandwidth correction of errors without resorting to MDS. The storage order is shown in Fig.7. Notice all these locations are modulo n since our Node sequence is circularly arranged. Hence, we assume that we have n nodes in total. The node structure details is as follows::

$$\bar{z}_0 = \begin{bmatrix} x_0 \\ y_1 \\ x_2 \boxplus y_2 \end{bmatrix}, \bar{z}_1 = \begin{bmatrix} x_1 \\ y_2 \\ x_3 \boxplus y_3 \end{bmatrix} .. \bar{z}_{n-1} = \begin{bmatrix} x_{n-1} \\ y_{\langle n \rangle_n} \\ x_{\langle n+1 \rangle_n} \boxplus y_{\langle n+1 \rangle_n} \end{bmatrix} \tag{3}$$

For an arbitrary node m, $0 \leq m \leq n - 1$,

$$\bar{z}_m = \begin{bmatrix} x_m \\ y_{\langle m+1 \rangle_n} \\ x_{\langle m+2 \rangle_n} \boxplus y_{\langle m+2 \rangle_n} \end{bmatrix} \equiv \begin{bmatrix} x_m \\ y_{\langle m+1 \rangle_n} \\ s_{\langle m+2 \rangle_n} \end{bmatrix} \tag{4}$$

Equation (4) is a simple fast regenerative code capable of burst error correction without resorting to MDS.

3.2 Node Correction fo Storage as a Service (SaaS)

We now illustrate our proposed failure nodes detection. As shown in Fig. 8, after we construct our distributed node storage on the cloud, database errors, memory errors, and node failures occur. Further causation is also due to hardware malfunctions, power outages, and database maintenance. Our goal is rapid, pervasive correction of failed nodes, independent of the cause. After node error detection, we generate a list of failed nodes in the set of n, defining binary 1 as the error nodes and binary 0 as operational nodes with no errors. Hence, we maintain a state-error vector to delineate the complete set of failed and correctly operating nodes. The length of the vector is the number of the nodes, n. In general, for a detection vector $l^{1 \times n}$, if node i has failed, then $l_{1,i} = 1$, otherwise it is 0. Thus,

$l^{10\times 1} = [1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0]$
indicates a 10 nodes system with 3 detected node errors or failures, nodes 0, 3, and 5.

3.3 Directly Fast Bandwidth Reparable

Typically, if errors occur in particular patterns, we can directly use the *Fast Bandwidth Reparation* to correct the failed nodes, without invoking the higher order regeneration capability of our more powerful MDS code. For an (n, k) MDS code with distributed storage across n nodes, we can fast bandwidth correct $\mathrm{u} = \mathrm{floor}\left(\frac{\mathrm{n}}{3}\right)$ nodes in parallel. To test whether we are able to use the Direct Fast Bandwidth Reparation, we propose the application of a fast-bandwidth detection matrix $\mathrm{T}^{\mathrm{n}\times\mathrm{n}}$ in the cloud application.

As shown in Fig.9, we receive a detection vector $l^{1\times n}$ from the cloud system. Then we measure the weight of the vector $l^{1\times n}$. If the $weight(l^{1\times n}) < u$, there exists a high probability that errors can be corrected using the fast repair procedure.

$$t_0^{n\times 1} = [1 \quad 1 \quad 1 \quad 0 \quad 0 \quad \dots \quad 0]^T(\text{Original})$$
$$t_1^{n\times 1} = [0 \quad 1 \quad 1 \quad 1 \quad 0 \quad \dots \quad 0]^T(\text{Shift } 1)$$
$$t_2^{n\times 1} = [0 \quad 0 \quad 1 \quad 1 \quad 1 \quad \dots \quad 0]^T(\text{Shift } 2)$$
$$\mathrm{t}_3^{\mathrm{n}\times 1} = [0 \quad 0 \quad 0 \quad 1 \quad 1 \quad \dots \quad 0]^{\mathrm{T}}(\text{Shift } 3)$$

Our testing matrix is defined by

$$T^{n\times n} = [t_0^{n\times 1} \quad t_1^{n\times 1} \quad t_2^{n\times 1} \quad \dots \quad t_{n-3}^{n\times 1} \quad t_{n-2}^{n\times 1} \quad t_{n-1}^{n\times 1}]^T,$$
That is,

$$\boldsymbol{T}^{n\times n} = \begin{bmatrix} 1 & 1 & 1 & 0 & \dots & 0 \\ 0 & 1 & 1 & 1 & \dots & 0 \\ 0 & 0 & 1 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \dots & \vdots \\ 0 & \dots & 0 & 1 & 1 & 1 \\ 1 & \dots & 0 & 0 & 1 & 1 \\ 1 & \dots & 0 & 0 & 0 & 1 \end{bmatrix} \qquad (5)$$

We define $\boldsymbol{M} = T^{n\times n} \times l^{n\times 1}$. Iff all the elements in $\boldsymbol{M}$ are either 1 or 0, then it is reparable.

If the detection vector is directly fast bandwidth reparable, as shown in Fig. 5, we invoke the Fast Bandwidth Algorithm to correct the error and regenerate the error-free nodes. Suppose node m has failed. A new matrix named $\boldsymbol{N_m}$ is built:

$$\boldsymbol{N}_m = [\bar{z}_{m-2} \quad \bar{z}_{m-1} \quad \bar{z}_{m+1} \quad \bar{z}_{m+2}]^T \qquad (6)$$

That is,

$$\boldsymbol{N}_m = \begin{bmatrix} x_{\langle m-2\rangle_n} & y_{\langle m-1\rangle_n} & x_{\langle m\rangle_n} + y_{\langle m\rangle_n} \\ x_{\langle m-1\rangle_n} & y_{\langle m\rangle_n} & x_{\langle m+1\rangle_n} + y_{\langle m+1\rangle_n} \\ x_{\langle m+1\rangle_n} & y_{\langle m+2\rangle_n} & x_{\langle m+3\rangle_n} + y_{\langle m+3\rangle_n} \\ x_{\langle m+2\rangle_n} & y_{\langle m+3\rangle_n} & x_{\langle m+4\rangle_n} + y_{\langle m+4\rangle_n} \end{bmatrix} \qquad (7)$$

From the $\boldsymbol{N_m}$ matrix, we know the repair of one failed node, $\boldsymbol{z_m}$, requires that access $\boldsymbol{z_{m-2}}$, $\boldsymbol{z_{m-1}}$, $\boldsymbol{z_{m+1}}$, $\boldsymbol{z_{m+2}}$. Since we have n nodes in total, we define a matrix N as:

$$\mathbf{N}^{\mathrm{n}\times\mathrm{n}} = \begin{bmatrix} \mathrm{N}_0 & & & \\ & \mathrm{N}_1 & & \\ & & \ddots & \\ & & & \mathrm{N}_{\mathrm{n}-1} \end{bmatrix} \qquad (8)$$

Figure 9 shows the entire system procedure for a fast bandwidth application applied to a Big Data source. After we encode our information source, we place it in the cloud using Storage as a Service (SaaS) paradigm. Storage nodes fail randomly. We apply a polling procedure for pervasive node detection. When node errors are detected, their locations are recorded and communicated to our correction system "SaaS Database." The vector, l, passed formulate a binary rule that determines whether we can apply Direct Fast Bandwidth repair or not. If yes, we execute the algorithm. If not, we use MDS pre-correction before we run the fast algorithm.

4 Conclusion

We have formulated an innovative cloud application that operates on Big Data in a distributed processing framework, using Map-reduce, while simultaneously operating at the optimum fast bandwidth rate. In Big Data storage systems, there is more than one node that can fail. In this case, we illustrated an optimal joint correction via maximally distance seperable codes and simple regenerative fast bandwidth codes in the cloud. Thus, severe error patterns that cannot be repaired by the simple regnerative code can be correct by the theoretical optimal MDS code. We also simulate the bandwidth performance and that it outperforms the prior methods of [8]. Moreover, we propose an innovative, patented, map-reduce algorithm ideal for fast bandwidth repair in cloud data center storage.

Table 1. Cloud Application Parmeters

	Param.	Description
Inputs to Cloud App.	$\mathcal{A}$	Big memory source, a 1 Exabyte Big Data-set
	λ	Poisson error arrival rate in bits/sec/512 bytes of mem.
	k	MDS code rate, k/N
Run Time Parameters	P	Number of parallel databases, MDS system, that used as the parallel Hadoop Map-Reduce system
	N	The number of nodes in MDS system that used as the parallel Mapping for Hadoop Map-Reduce
	M	Node size memory in MDS system
	Tper	Time step period in seconds at which data bases errors are detected and memory is reconstructed
	(T, T x L)	(Hadoop Fast bandwidth correction latency, MDS correction latency is $L = N/3 + 2N^3/9$ times slower)

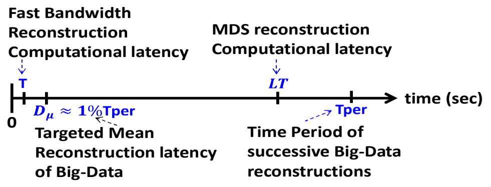
Fast Bandwidth Reconstruction Computational latency
MDS reconstruction Computational latency
T
LT
time (sec)
0
$D_\mu \approx 1\%$Tper
Tper
Targeted Mean Reconstruction latency of Big-Data
Time Period of successive Big-Data reconstructions

Figure 1. Timeline Latency Model

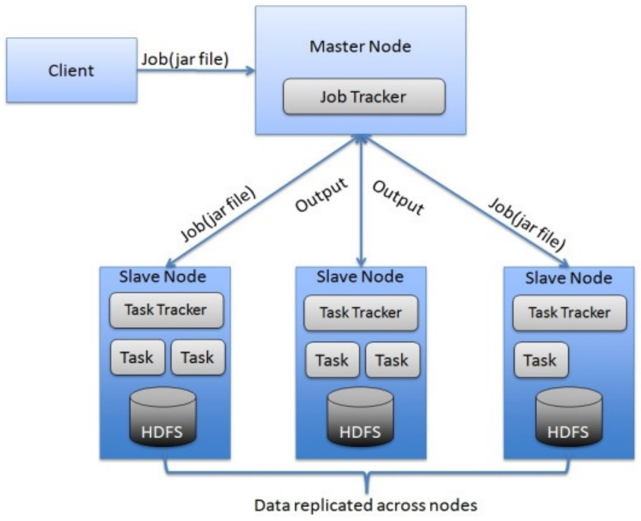

Figure 2. System Architecture- FMR code is implemented in Task Tracker and the dynamic configuration parameters are communicated from Name Node to Data Nodes through Job Tracker.

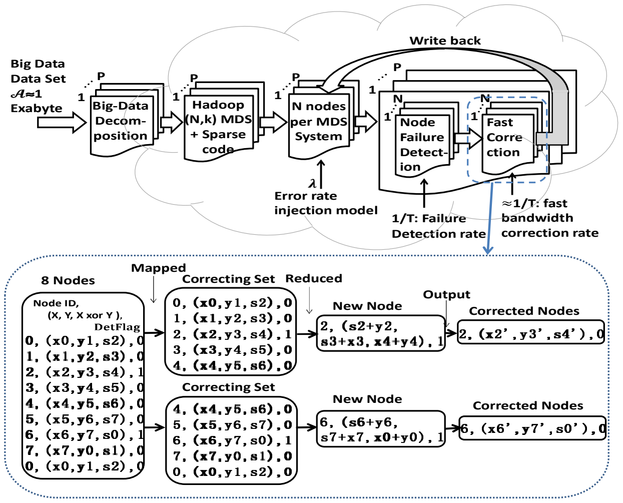

Figure 3. Cloud encode and reconstruction procedure for a given Big Data size, $\mathcal{A}$, and error arrival rate, $\boldsymbol{\lambda}$, into memory.

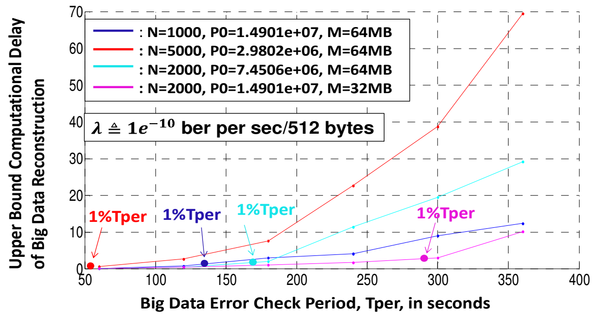

Figure 4. Upper Bound Computational Latency of Big Data Reconstruction Versus Big Data Error Check Period, Tper.

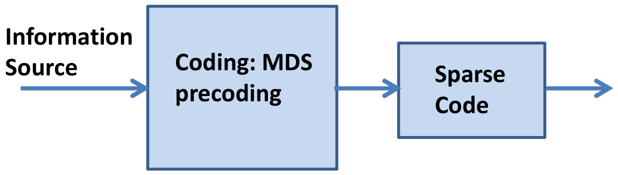

Figure 5. MDS Pre-coding and Sparse Code construction

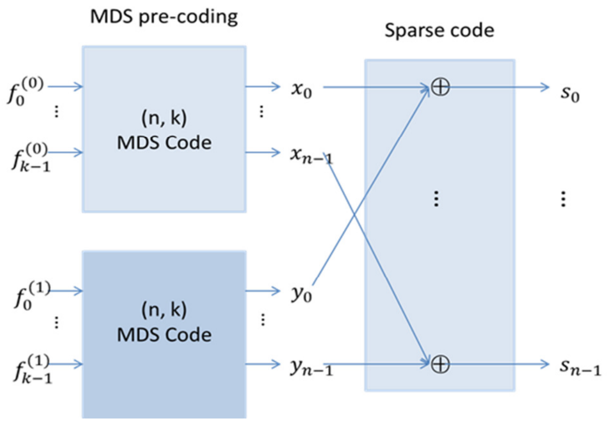

Figure 6. Example of (n,k) SRC coding.

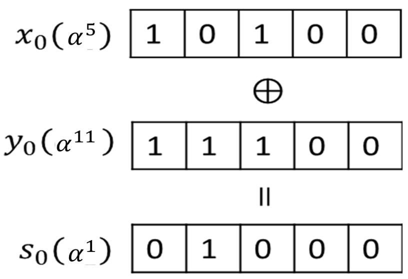

Figure 7. Example of finite field addition.

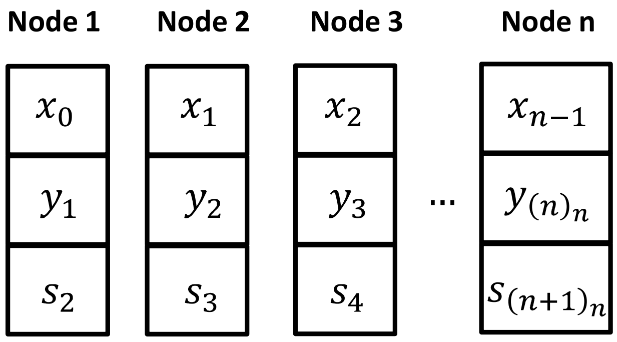

Figure 8. (n,k) SRC storage nodes placement [9]

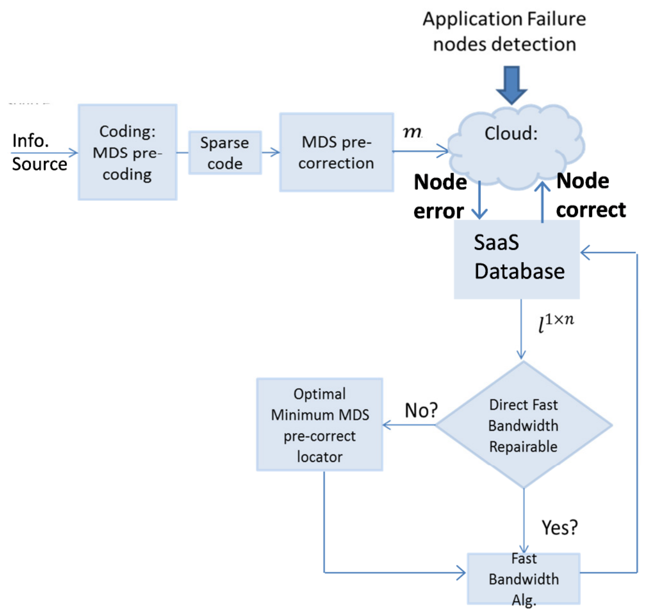

Figure 9. Fast Big Data Bandwidth Algorithm

References

[1] A. Hadoop. Available: http://hadoop.apache.org/

[2] J. Venner, *Pro Hadoop*, 2009.

[3] H. Weatherspoon and J. Kubiatowicz, "Erasure Coding Vs. Replication: A Quantitative Comparison," presented at the Revised Papers from the First International Workshop on Peer-to-Peer Systems, 2002.

[4] K. Shvachko, K. Hairong, S. Radia, and R. Chansler, "The Hadoop Distributed File System," in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, 2010, pp. 1-10.

[5] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Commun. ACM,* vol. 51, pp. 107-113, 2008.

[6] O. O. S. C. C. Software. Available: http://www.openstack.org/

[7] G. Wang and Y. Zhao, "A Fast Algorithm for Data Erasure," ISI 2008 IEEE International Conference on Intelligence and Security Informatics, pp. 245-256, 2008.

[8] D.S. Papailiopoulos, J. Luo, A.G. Dimakis, C. Huang, and J. Li, "Simple Regenerating Codes: Network Coding for Cloud Storage," *The 31st Annual IEEE International Conference on Computer Communications: Mini-Conference,* pp. 2801-2805, 2012.

[9] W. Ailan, L. Yunqiang, Z. Xiaoyong, "Analysis of Corresponding Structure of Differential Branch of MDS Matrixes on Finite Field," *2010 Third International Conference on Intelligent Networks and Intelligent Systems*, pp. 381-384, 2010.

[10] Y. Shang, D. Wang, X. Xia, "Flexible Signal Space Diversity Techniques From MDS Codes With Fast Decoding," *2010 Proceedings IEEE Globecom*, pp. 1-5, 2010.

[11] Brian Kelley and Xue Qin, Fast Bandwidith Reconstruction of Big Data Sets in a Cloud Computing Environment Using a Parallel Map Reduce Framework, Provisional Patent, serial number , 62/090,868, official filing date 12/11/2014.

[12] Xue Qin, A Fast Map Reduce Algorithm for Exact-Repair Reconstruction of Big-Data in Cloud Storage, MS Thesis, University of Texas at San Antonio, Dec. 2014.

[13] Y. Itoh, M. Momodomi, R. Shirota, Y. Iwata, R. Nakayama, R. Kirisawa, *et al.*, "An experimental 4 Mb CMOS EEPROM with a NAND structured cell," in *Solid-State Circuits Conference, 1989. Digest of Technical Papers. 36th ISSCC., 1989 IEEE International*, 1989, pp. 134-135.