ISSN No: - 2456 - 2165

# An Efficient Regenerating Coding for Recovery Single and Concurrent Failures in Hadoop

Mr.T.Karthikeyan /AP / CSE / Knowledge Institute of Technology, Salem

N.Vanitha / II-M.E / CSE / Knowledge Institute of Technology, Salem

**ABSTRACT :- Data availability is perilous in distributed storing systems, especially when node failures are prevalent in real life. A key requirement is to minimize the amount of data transferred among nodes when recovering the lost or unavailable data of failed nodes. The retrieval solutions for proposed system based on regenerating codes, which are shown to provide fault-tolerant storage and minimum recovery bandwidth. Existing optimal regenerating codes are designed for single node failures. The failures in distributed storing systems, which supplements existing optimal regenerating codes to support a general number of failures including single and concurrent failures. The distributed storage system achieves single and concurrent failures having minimum possible recovery bandwidth for most cases. The proposed system implements single and concurrent failures in distributed storage systems and evaluate our prototype atop a Hadoop HDFS cluster tested with up to 2 storage nodes. The experimental result shows the single and concurrent failures in distributed storage systems prototype conforms to our theoretical findings and achieves recovery bandwidth and stores the lost data.**

***Keywords:*** *Single and Concurrent failures, Erasure coded System, Recovery bandwidth, Mapreduce* **.**

## I. INTRODUCTION

To decrease storage overhead, cloud file methods are transitioning from repetition to erasure codes. This process has revealed new dimensions on which to evaluate the performance of different coding schemes: the amount of data used in recovery and when performing degraded reads. To ensure data availability, erasure codes have been widely deployed in large-scale storage systems. The proposed algorithm that finds the optimal number of code word symbols needed for recovery for any XOR-based erasure code and produces recovery schedules that use a minimum amount of data. The proposed algorithm explores recovery solutions based on regenerating codes algorithm, which are shown to provide fault-tolerant storage and minimum recovery bandwidthkey feature of simultaneous failure revealing is that it maintains existing optimal regenerating code constructions and the underlying regenerating-coded data. A new recovery scheme stops existing regenerating codes. This paper makes the following contributions. The proposed system achieves the minimum recovery bandwidth for a majority of concurrent failure patterns.

### A. *Hadoop Overview*

When data sets go beyond a single storage capacity, it is necessary to distribute them to multiple independent computers. Trans-computer network storage file management system is called distributed file system. HDFS cluster configuration is simple.

Hadoop is an open source framework, from the Apache foundation, capable of processing large amounts of heterogeneous data sets in a distributed fashion across clusters of commodity computers and hardware using a simplified programming model. Hadoop has two core components: HDFS and Map Reduce.

### *B. HDFS*

Hadoop Distributed file system proposals a highly dependable and distributed storage, and ensures consistency, even on commodity hardware, by duplicating the data across multiple nodes. This ensures high accessibility and fault tolerance.

**Name Node** holds the data about all the other nodes in the Hadoop Cluster, files present in the cluster, constituent blocks of files and their locations in the cluster, and other data useful for the action of the Hadoop Cluster.

**Data Node** is in charge for holding the data.

### *C. Map Reduce*

Map Reduce offers an analysis system which can perform complex computations on large datasets. This component is responsible for performing all the computations and works by breaking down a large complex computation into multiple tasks and assigns those to individual worker/slave nodes and takes care of coordination and consolidation of results.

**Job Tracker**retains track of the individual jobs assigned to each of the nodes and match up the exchange of information and results.**Task Tracker** is responsible for running the task / computation assigned to it.

### *D. Erasure Coded System*

Erasure code is the underlying component used by BlockFixer and RaidNode to generate parity blocks and to fix parity/source blocks. Erasure Code does encode and decode. When encoding, Erasure Code takes several source bytes and generate some parity bytes.

## II. SYSTEM ANALYSIS

### A. *Existing System*

Most of previous works still restrict their attention on the stripe-level recovery for single failures, but miss a fact that the real storage systems usually rotationally map the logical disks to physical disks in order to alleviate the parity disks from being the hot point when suffering from a huge number of writes. Though Luo's schemes well speed up the process of single disk failure recovery in the stripe-level, it is still unknown which method is the best choice once moving to the scenario where logical disks are mapped rotationally.
The conventional method to recover from single failures is to select k surviving disks and create a kw-element decoding bit matrix from the corresponding rows of generator matrix. The product of the wk elements (in the k surviving disks) and the converted decoding bit matrix will generate the original wk data elements. In recent years, some researches have been proposed for improving the recovery speed. In this section, we introduce these works.

*Recovery Equations:* A recovery equation is composed by a series of data elements and parity elements, whose XOR sum equal to zero. If one element of the recovery equation is lost, we can reconstruct it by other survived elements. The existing a hybrid recovery method for RDP code. Thanks to the overlapping elements, this method reduces up to 25 percent I/O cost than the conventional method.

*Hybrid Recovery Methods:* The existing a similar method to minimize recovery I/O cost in EVENODD code optimal method for X-Code and have some extend investigate in stack-based recovery. However, this work require the disks to rotate by following the fixed method they proposed and actually this rotated method is rarely used in real systems.

Moreover, this method is only applicable for X-Code. General search-based algorithms to generate recovery schemes (C-Scheme and U-Scheme). Similar to Khan's Scheme, C-Scheme searches the solution with the minimal I/O cost, but it has an extra condition that the read accesses on the heaviest loaded disk are the minimal.

### B. *Problem Identification*

- Single disk failure in a storage system that uses double faut tolérant array codes
- High throughput on the total disk reconstruction
- High Computation cost of triple construction & Re-configurability.
- More replication data and Single disk failure recovery is an expensive task
- It can be incorporated to known RAID techniques
- Recovery time and affects the system service performance.
- Each row to recover each lost symbol of the failed disk so it will take some time.

### B. *Proposed System*

The propose system which chains both single and concurrent failure retrieval and aims to minimalize the bandwidth of recovering a general number of failures. The proposed system augments existing optimal regenerating code constructions, which are designed for single failure recovery, to also support concurrent failure recovery. A main feature of is that it holds existing optimum regenerating code constructions and the primary regenerating-coded data. That is, adds a new recovery scheme atop existing regenerating codes. This paper makes the following contributions. We theoretically show that achieves the minimum recovery bandwidth for a majority of concurrent failure patterns. We also propose extensions to CORE to achieve sub-optimal bandwidth saving even for the remaining concurrent failure patterns. We implement and experiment prototype on a Hadoop Distributed File System (HDFS) testbed with up to 20 storage nodes. We show that compared to erasure codes, achieves recovery throughput gains with up to 3.4× for single failures and up to 2.3× for concurrent failures.

An erasure coded storage system is also partitioned into stripes, which are maximal sets of disk blocks that are independent on each other in terms of redundancy relations. Each block is partitioned into a fixed number of elements, which are fixed-size units of data or parity information and the number is denoted as w. We label the w elements on the I th data disk as on the ith parity disk. Generator bit matrix for erasure array codes. All erasure array codes can be represented by the generator matrix product. We show an example of Cauchy Reed-Solomon code, the kr data elements are organized as a kr-element bit vector, while the generator matrix is a wn x wk bit matrix. Based on the generator bit matrix, we can easily compute all the parties, i.e., computing the parity information.

### C. *Architecture*

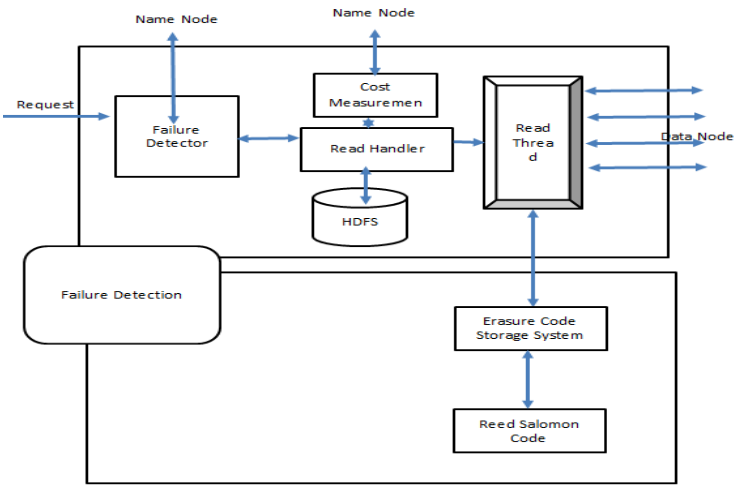


Figure: 1 Architecture diagram

The diagram illustrates the process of failure detection recovery. It reads the number of file request from Hadoop distributed file system. If the file is not establishing failure detector then it assigns the read handler. The read handler calls the erasure code storage system with read Salomon code to generate the missing file data. Cost measurement helps for measuring the number of cost from different data node. Finally select the minimum number of cost of data node**.**

## III. MODULES

### A. *Erasure Coded Storage System*

Erasure coded storage systems enhancedismissal for fault tolerance. Specifically, a system of n disks is partitioned into k disks that hold data and m disks that hold coding information. The coding information is considered from the data using an erasure code.

*Two Properties:* First, it must be Maximum Distance Separable (MDS), which means that if any m of the n disks fails, their contents may be recomputed from the k surviving disks. Second, it must be systematic, which means that the k data disks hold un-encoded data.

Building blocks the MSR code constructions. Our observation is that any optimal MSR code construction can be defined by two functions. Let $Enc_{i,j}$be the encoding function that is called by node $N_i$ to generate an encoded symbol for the failed node $N_j$ using the r = n − k stored symbols in node Ni as inputs; let $Rec_{i,j}$ be the reconstruction function that returns the set of ns − k stored symbols of a failed node Ni using the encoded symbols from the other n−1 surviving nodes as inputs.

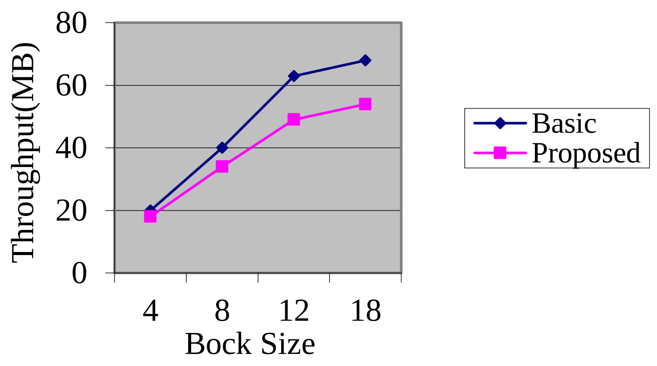


Figure: 2. Comparison of different input block size

### B. *Failure Detection*

The proposed express the virtual symbols as a function of real symbols by solving a system of equations. Conversely, we annotation that for certain failure patterns (i.e., the set of failed nodes), the scheme of equations cannot return a unique solution. A failure pattern is said to be decentif we can exceptionally express the virtual symbols as a function of the real symbols, or corrupt. Our aim is to diminish the recovery bandwidth even for immoral failure patterns. We now extend our baseline approach of proposed to deal with the bad failure patterns, with an objective of reducing the recovery bandwidth over the conventional recovery approach.

We evaluate the recovery performance. For a given (n, k), we construct our HDFS testbed with n Data Nodes, one of which also organizes the Raid Node for striping the encoded data.

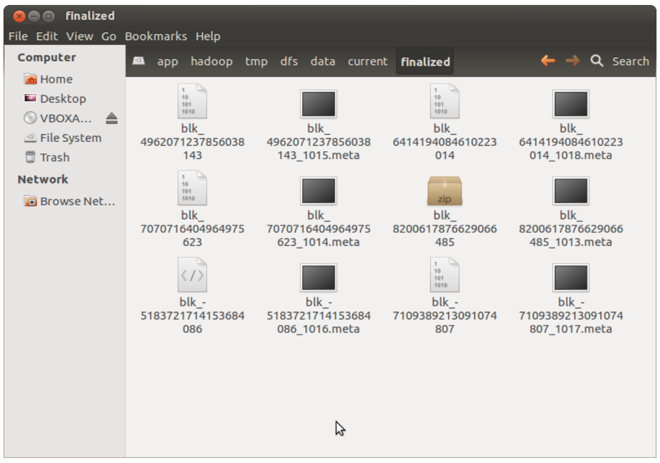
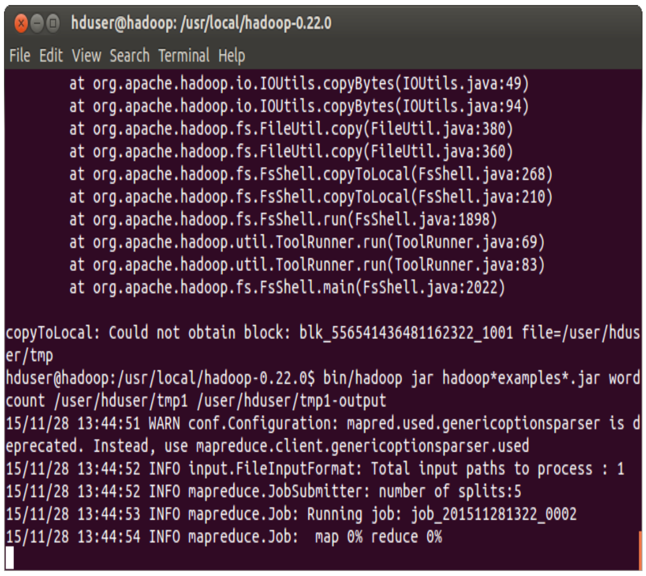

finalized
File Edit View Go Bookmarks Help
Computer
Home
Desktop
VBOXA...
File System
Trash
Network
Browse Net...
app hadoop tmp dfs data current finalized
Search
blk_ 4962071237856038 143
blk_ 4962071237856038 143_1015.meta
blk_ 6414194084610223 014
blk_ 6414194084610223 014_1018.meta
blk_ 7070716404964975 623
blk_ 7070716404964975 623_1014.meta
zip
blk_ 8200617876629066 485
blk_ 8200617876629066 485_1013.meta
blk_- 5183721714153684 086
blk_- 5183721714153684 086_1016.meta
blk_- 7109389213091074 807
blk_- 7109389213091074 807_1017.meta

Figure: 3. Failure detection numbers of files

### C. *Map/Reduce Jobs*

Map Reduce job is collected of four parts: Setup, Map, Reduce, and Cleanup, among which only the task of Map includes, degraded reads. Therefore, mainly improves the Map tasks. It also brings benefits for execution of Reduce tasks as the Map tasks can return the intermediate results fast

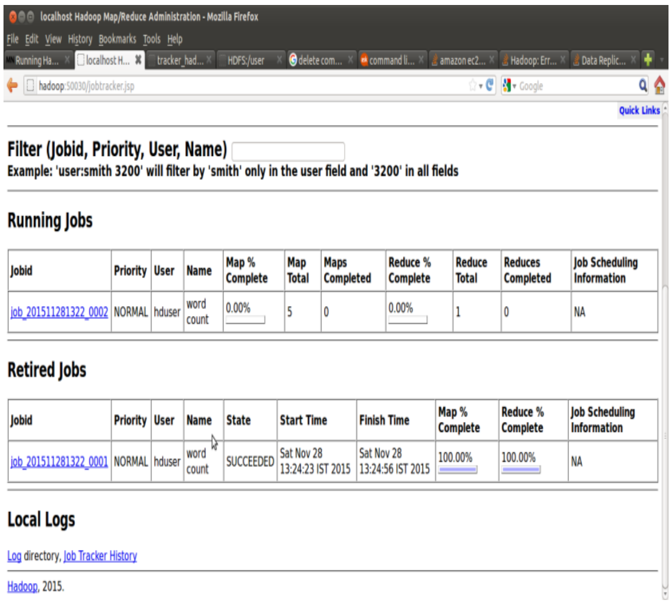

hduser@hadoop: /usr/local/hadoop-0.22.0
File Edit View Search Terminal Help

```
        at org.apache.hadoop.io.IOUtils.copyBytes(IOUtils.java:49)
        at org.apache.hadoop.io.IOUtils.copyBytes(IOUtils.java:94)
        at org.apache.hadoop.fs.FileUtil.copy(FileUtil.java:380)
        at org.apache.hadoop.fs.FileUtil.copy(FileUtil.java:360)
        at org.apache.hadoop.fs.FsShell.copyToLocal(FsShell.java:268)
        at org.apache.hadoop.fs.FsShell.copyToLocal(FsShell.java:210)
        at org.apache.hadoop.fs.FsShell.run(FsShell.java:1898)
        at org.apache.hadoop.util.ToolRunner.run(ToolRunner.java:69)
        at org.apache.hadoop.util.ToolRunner.run(ToolRunner.java:83)
        at org.apache.hadoop.fs.FsShell.main(FsShell.java:2022)

copyToLocal: Could not obtain block: blk_556541436481162322_1001 file=/user/hdus
er/tmp
hduser@hadoop:/usr/local/hadoop-0.22.0$ bin/hadoop jar hadoop*examples*.jar word
count /user/hduser/tmp1 /user/hduser/tmp1-output
15/11/28 13:44:51 WARN conf.Configuration: mapred.used.genericoptionsparser is d
eprecated. Instead, use mapreduce.client.genericoptionsparser.used
15/11/28 13:44:52 INFO input.FileInputFormat: Total input paths to process : 1
15/11/28 13:44:52 INFO mapreduce.JobSubmitter: number of splits:5
15/11/28 13:44:53 INFO mapreduce.Job: Running job: job_201511281322_0002
15/11/28 13:44:54 INFO mapreduce.Job:  map 0% reduce 0%
```

Figure 4. Map/Reduce job Counts

Run three MapReduce applications: (i) WordCount, which computes the occurrence frequency of each word in the dataset; (ii) Dedup, which removes duplicate lines in the dataset and outputs all unique lines; and (iii) Grep, which extracts matching strings from text files and counts their occurrences.



Filter (Jobid, Priority, User, Name)
Example: 'user:smith 3200' will filter by 'smith' only in the user field and '3200' in all fields

Running Jobs

| Jobid | Priority | User | Name | Map % Complete | Map Total | Maps Completed | Reduce % Complete | Reduce Total | Reduces Completed | Job Scheduling Information |
|---|---|---|---|---|---|---|---|---|---|---|
| job_201511281322_0002 | NORMAL | hduser | word count | 0.00% | 5 | 0 | 0.00% | 1 | 0 | NA |

Retired Jobs

| Jobid | Priority | User | Name | State | Start Time | Finish Time | Map % Complete | Reduce % Complete | Job Scheduling Information |
|---|---|---|---|---|---|---|---|---|---|
| job_201511281322_0001 | NORMAL | hduser | word count | SUCCEEDED | Sat Nov 28 13:24:23 IST 2015 | Sat Nov 28 13:24:56 IST 2015 | 100.00% | 100.00% | NA |

Local Logs

Log directory, Job Tracker History

Hadoop, 2015.

Figure: 5. Running and retired Jobs

*D. Failure Recovery Pattern*

The express the virtual symbols as a function of real symbols by solving a system of equations. Conversely, we annotation that for some failure patterns (i.e., the set of failed nodes), the system of equations cannot return a unique solution. A failure pattern is said to be good if we can uniquely express the virtual symbols as a function of the real symbols, or bad otherwise. Our goal line is to decrease the retrieval bandwidth even for immoral failure patterns. The proportion of bad failure patterns is in general very small, with at most 0.9% and 1.6% for IA and PM codes, respectively.

The proposed algorithm recovery builds on the relayermodel, in which a relayer daemon coordinates the recovery operation. The depicts the relayer model. During recovery, each surviving node performs two steps: (i) I/O: it reads its stored data, and (ii) encode (for regenerating codes only): it combines the stored data into some linear combinations.

$$(\alpha_{MSR}, \gamma_{MSR}) = \left(\frac{\mathcal{M}}{k}, \frac{\mathcal{M}d}{k(d-k+1)}\right).$$

The repair bandwidth MSR = d_MSR is a decreasing function of the number of nodes d that participate in the repair.

<u>**Recovery Speed**</u>

| **Hadoop Technology** | **No of Disk** | | | |
|---|---|---|---|---|
| | 1 | 2 | 3 | 4 |
| **Basic** | 22 | 17 | 15 | 12 |
| **Proposed** | 28 | 25 | 20 | 19 |

Table: 1 Comparison of recovery speed for word count

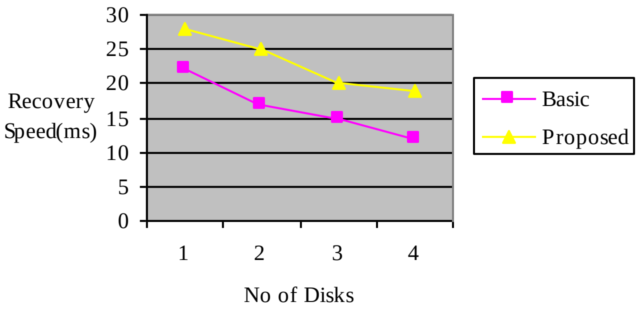


Figure 6. Comparison of recovery speed for word count

The above figure shows the comparison of number of input block size vs. reading throughput map/reduce. Compare to existing map/reduce block size evaluation the proposed system provide high throughput value of map/reduce.

<u>Recovery Time:</u>

| **Hadoop Technology** | **No of Reduce Task** | | | |
|---|---|---|---|---|
| | 1 | 2 | 3 | 4 |
| Basic | 54.31 | 48.56 | 45.38 | 38.98 |
| Proposed | 27.57 | 17.39 | 12.23 | 8.42 |

**Table: 2** Comparison of Time speed

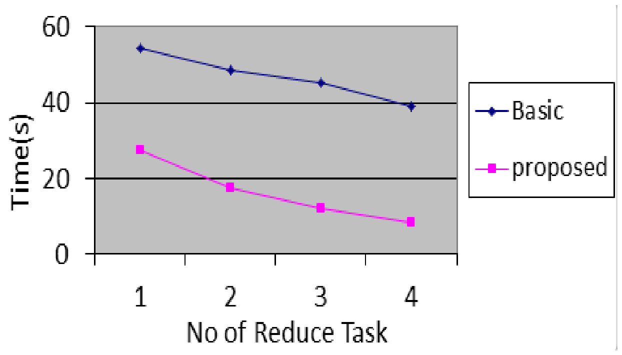


Figure 7. Comparison of Time speed

The above figures shows that the number reduce slot with different machine to calculating word count program time duration for existing and proposed system. The proposed system takes minimum amount of time for word count program.

## IV. CONCLUSION

The use of regenerating codes to provide fault tolerant storage and minimizing the bandwidth of data transfer during recovery. We propose a system which generalizes existing optimal single-failure-based regenerating codes to support the recovery of both single and concurrent failures. We theoretically show that CORE minimizes the reconstruction bandwidth in most concurrent failure patterns**.**

## V. FUTURE ENHANCEMENT

The future work plan to coding based recovery for single and concurrent failures in Distributed Storage Systems which augments existing optimal regenerating codes to support a general number of failures including single and concurrent failures. It theoretically shows that algorithm achieves the minimum possible recovery bandwidth for most cases. This method is that it retains existing optimal regenerating code constructions and the underlying regenerating-coded data. That is, algorithm adds a new recovery scheme atop existing regenerating codes. This makes the following contributions. We theoretically show that algorithm achieves the minimum recovery bandwidth for a majority of concurrent failure patterns. We also propose extensions to algorithm to achieve sub-optimal bandwidth saving even for the remaining concurrent failure patterns.

## REFERENCE

[1] Yingxun Fu, Jiwu Shu, ZhirongShen, and Guangyan Zhang "Reconsidering Single Disk Failure Recovery for Erasure Coded Storage Systems: Optimizing Load Balancing in Stack-Level," in Proc, IEEE Transaction on parallel and Distributed Systems., May 2016, pp. 1457-1469.

[2]Blaum, M., Bruck, J., and Nebib, J. "EVENODD: An efficient scheme for tolerating double disk failures in RAID architectures," IEEE Trans. Inf. Theory, vol. 45, no. 1, pp. 46–59, Jun. 1999.

[3] Corbett, P., English, B., Goel, A., Grcanac, T., Kleiman, S., Leong, J., and Sankar, S. "Row-diagonal parity for double disk failure correction," in Proc. 3rd USENIX Conf. File Storage Technol., San Francisco, CA, USA, Mar. 2004, p. 1.

[4] Fu, Y., Shu, J., and Luo, X. "A stack-based single disk failure recovery scheme for erasure coded storage systems," in Proc. IEEE 33rd Int. Symp. Rel. Distrib. Syst., Oct. 2014, pp. 136– 145.

[5] Greenan, K., Li, X., and Wylie, J., (2010) "Flat XOR-based erasure codes in storage systems: Constructions, efficient recovery, and tradeoffs," in Proc. 26th Symp. IEEE Mass Storage Syst, pp. 1–14.

[6] Huang, C. andXu, L.,(2005) "STAR: An efficient coding scheme for correcting triple storage node failures," in Proc. USENIX Conf. File Storage Technol., p. 15.

[7] Luo, X., and Shu, J., (2013) "Load-balanced recovery schemes for single disk failure in storage systems with any erasure code," in Proc. 42nd IEEE Int. Conf. Parallel Process., pp. 552–561.

[8] Luo, X., and Shu, J., (2013) "Load-balanced recovery schemes for single disk failure in storage systems with any erasure code," in Proc. 42nd IEEE Int. Conf. Parallel Process., pp. 552–561.

[9] Wang, Z., Dimakis, A., and Bruck, J., (2010) "Rebuilding for array codes in distributed storage systems," in Proc. IEEE GLOBECOM Workshops,pp. 1905–1909.

[10] Xu, S., Li, R., Lee, P., Zhu, Y., Xiang, L., Xu, Y., and Lui, J., (2013) "Single disk failure recovery for X-code-based parallel storage systems," IEEE Trans. Comput., vol. 63, no. 4, pp. 995–1007.

[11] Zhu, P.,Lee, Y., Hu, Y., Xiang L., and Xu, Y., (2012) "On the speedup of Single-disk failure recovery in XOR-coded storage systems: Theory and practice," in Proc. 28th IEEE Mass Storage Syst. Technol., pp. 1–12.