Indian Journal of Science and Technology, Vol 9(30), DOI: 10.17485/ijst/2016/v9i30/99067, August 2016
ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

Adversarial Effect in Distributed Storage Systems

C. K. Shyamala* and R. R. Sharanya

Department of Computer Science and Engineering, Amrita School of Engineering, Amrita Vishwa Vidyapeetham (University) Amrita Nagar P.O, Ettimadai, Coimbatore 641112, Tamil Nadu, India; ck_shyamala@cb.amrita.edu, Sharanyarr93@gmail.com

Abstract

Objectives: The critical problem of mass storage and access is handled by Distributed Storage System (DSS) by incorporating data Replication and/or Dispersal techniques. Handling the hybrid failures is very crucial for the design of reliable and efficient DSS. The realistic combination of Crash and Non-Crash faults in DSS disallow fail stop assumption in design. This paper provides an insight into different types of failures, adversaries, proposes a systematic scheme to simulate adversarial effect in DSS. **Methods/Statistical Analysis:** Information Dispersal Algorithm (IDA) and Reed Solomon codes (RS) are the methods used to analyze the performance of the system. **Findings:** The performance of the system and robustness of the RS Coded DSS is evaluated and analyzed for various Workload, RS (n,k) values, % of Hybrid failures to design reliable and efficient Distributed Storage System. **Application/Improvement:** Reed Solomon codes improves the performance of the system by handling both errors and erasures compared to Information Dispersal Algorithm.

Keywords: Adversary, Dispersal, Distributed Storage System (DSS), Hybrid failures, Replication, RS Codes

1. Introduction

The volume of data is growing exponentially; the logical solution to handle voluminous data is Distributed Storage System (DSS). Replication and/or Dispersal are employed to provide fault tolerant store and access architectures in DSS. The primary goal of DSS is to provide Availability, Fault tolerance and Scalability. Node failures is inevitable in DSS; where hybrid failures is the combination of Crash and Non-Crash faults. This is an important factor to be considered while designing reliable DSS, guaranteeing Availability, Fault tolerance and Scalability. The presence of hybrid failures in DSS open up investigations into the role and impact of Adversary in designing DSS architecture. This paper provides an insight into hybrid failures and their impact on storage. A simulated environment enabling systematic injection of hybrid failures for distributed storage and access is the focus of this paper. DSS for RS based dispersal is analyzed in the presence of both Crash and Non - Crash faults.

In a distributed environment, a standard approach for data storage is replication where multiple copies of a file (information) are stored at several nodes to provide fault tolerance. This increases the bandwidth consumption and storage efficiency is declined[1-5]. The alternative choice to provide efficient storage at low cost is Dispersal technique. In Dispersal, data is splitted into k fragments and parity is added to obtain n fragments by using dispersal techniques like RS Codes[6-8] IDA[9-11]. These fragments are sent to several nodes where each node is having unique fragments and can be reconstructed if k out of n fragments is obtained. In case of network partitioning, sites are partitioned and the partitions may not be able to communicate with each other. Quorum-based technique is the best solution for this problem. When there is a simultaneous read and write request the availability is at risk. In spite of the availability issues the fault tolerance capability of quorum system is remarkable[12]. Research of DSS particularly in Quorum based approaches[13-15] assume Fail Stop environment. i.e, when a node fails it does not send data to other nodes. This assumption is rather unrealistic in designing DSS's that focus on improving availability, scalability, and fault tolerance. This implies that the operating environment is fully trusted and devoid of adversary. Such an assumption being far-fetched, the need for considering the role and impact of adversary in DSS design

*Author for correspondence

prompts investigation of adversaries. To assess the impact of fault tolerant techniques at DSS there is a rising need to experiment in the presence of failures. The paper focuses on proposing a model for a systematic injection of failures in DSS.

2. Fault Tolerant Design

The study of role and impact of hybrid failures for DSS techniques of replication and dispersal demands modelling of crash and non-crash faults and formulation of a simulation scheme to inject hybrid failures at storage. The proposed work provides a detailed simulation scheme facilitating the configuration for total number of storage nodes (n), (n,k) dispersal (when k = 1, equivalently replication), R/W workload, % of malicious effect (non-crash), % of crash effect. The simulation scheme enable failure injection in the DSS under consideration. The robustness of RS codes (dispersal technique) in handling the effects of adversary in DSS-dispersal is brought out. The importance of designing DSS to tolerate both errors & erasures in the presence of adversary is highlighted. The experimentation is performed in a cluster environment using MPI.

2.1 Dispersal Techniques

To provide fault tolerance, either Replication or Dispersal techniques can be used in DSS. The shortcomings of Replication technique is when an adversary gets access to any one of the replica, the data is available and security (confidentiality) is lost. When Dispersal technique is used instead of replication, the adversary needs access to k servers to get the original data which is highly difficult and increases the security (confidentiality). To provide better security, Dispersal technique is suggested in this work. Information Dispersal Algorithm (IDA), Reed Solomon (RS) codes are examples of dispersal techniques. Comparison of these two techniques is done and is found that RS code outperforms IDA. IDA fails to correct the errors in the data and it can tolerate only erasure up to n-k. The data which is unavailable or missing is termed as erasure. Let, the message transmitted be 10010100100 and the received message be 10_1010_1_0. In addition to erasure at positions 3,8,10 RS codes are capable of tolerating errors up to (n-k)/2; So RS codes standout as the most logical choice for dispersal at DSS. Dispersal using RS Code is illustrated in Figure 1.

Now mapping these for server faults and failures at DSS we have the following,

- A server may be available but may hold and provide incorrect data because of adversary control, H/W error, CDRAM, CPU errors and the like as illustrated in Figure 2. This part of data is equivalently errors in coding, as it is available for Read/Write operation but is incorrect.
- A server may be unavailable because of crash, N/W partition & the like as illustrated in Figure 3. This part of data is equivalently erasures in coding, as it is not available for Read/Write operation performed in DSS.

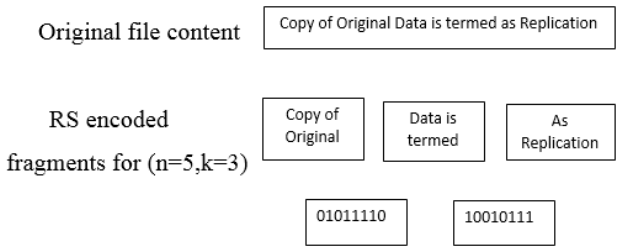

Figure 1. Dispersal using RS Codes

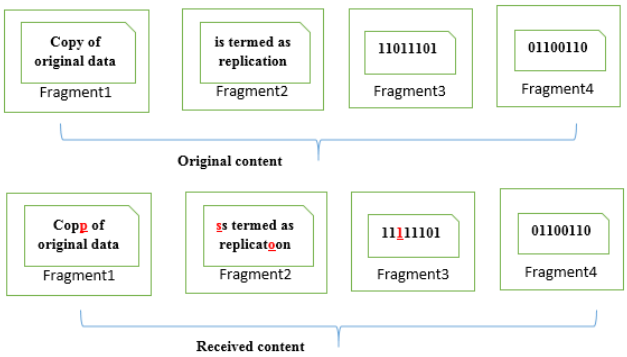

Figure 2. Error

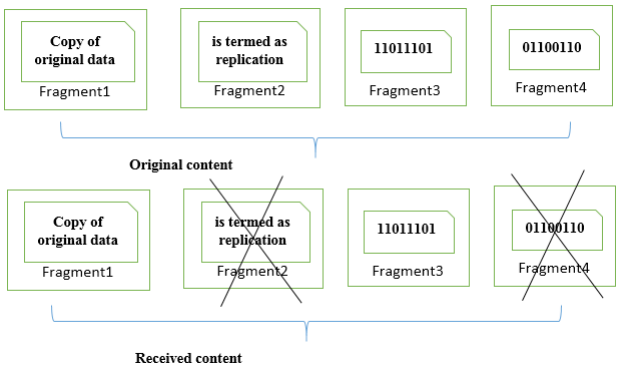

Figure 3. Erasure

3. Simulation Scheme

3.1 Failure Injection

To analyze the performance of DSS in the presence of adversary, failure injection is designed and simulated to enable the injection of Crash and Non-crash faults in the Distributed Storage System. The nodes can be classified as crashed and non-crashed nodes. Out of non-crash nodes, 2 possibilities arise. One is nodes are available but corrupted another chance is nodes are available and correct. The nodes which are available and corrupted is controlled by the adversary. It may be any type of adversary as discussed in[16,17]. Series of test cases will be generated as a result of this simulation to study the performance and behavior of the DSS in presence of adversary.

3.2 Configuration

MPI cluster has been setup using 3 systems having Ubuntu version- 14.04 64 bit OS, Open MPI version - 1.8.4. Adversary model have been simulated based on the configuration details to visualize the behavior of different adversaries present in the DSS. Initial configuration details are specified in Table 1. The configuration parameters are explained in Table 2. The status of nodes at read and writes operations are explained in Table 3.

Table 1. Initial Configuration

Case	No. of. Runs	Type of workload	RS (n,k)	% of Hybrid failures
1	2500	Equal Read & write	(10,6)	25%
2	4000	Read dominated	(14,10)	50%
3	5500	Write dominated	(6,3)	75%
4	7000	Write dominated	(255,223)	50%
5	8000	Read dominated	(255,239)	50%

Table 2. Configuration Parameters

Parameter	Description
n	Total no. of. servers
k	Minimum no. of. servers required
RS (n,k)	Dispersal code
r	Read dominated
w	Write dominated
r-w	Equal read & write
hf	% of Hybrid Failure

Based on the (n,k) scheme in the configuration table, files are fragmented. Different test case has been generated to analyze the performance of DSS in presence of adversary and the Robustness of RS codes are realized.

3.3 Test Cases

Case 1-A: No. of. Runs = 2500, workload = Equal Read and Write, RS (n, k) = (10, 6), % of Hybrid failures = 25% and the status of nodes for case 1-A can be referred in Figure 4.

Table 3. Node Status at Read and Write operations

Type	Availability and Correctness	Abbreviation
hf-1	Not available for any operation	NA-R/NA-W
hf-2	Available with erroneous data for the specific operations	AE-R/AE-W
hf-3	Available persistently with erroneous data	APE-R/APE-W
hf-4	Available with correct data	AC_R/AC_W

case 1-A runs=2500				Equal workload		% =25					
Run#	Workload	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
1	Write	AC_W	NA-W	APE-W	NA-W	AC_W	AC_W	NA-W	AC_W	AC_W	AC_W
2	Write	AC_W	NA-W	APE-W	NA-W	AC_W	AC_W	NA-W	AC_W	AC_W	AC_W
3	Read	AC_R	NA-R	APE-R	NA-R	AC_R	AC_R	NA-R	AC_R	AC_R	AC_R
4	Write	AC_W	NA-W	APE-W	NA-W	AC_W	AC_W	NA-W	AC_W	AC_W	AC_W
5	Read	AC_R	NA-R	APE-R	NA-R	AC_R	AC_R	NA-R	AC_R	AC_R	AC_R
6	Read	AC_R	NA-R	APE-R	NA-R	AC_R	AC_R	NA-R	AC_R	AC_R	AC_R
7	Read	AC_R	NA-R	APE-R	NA-R	AE-R	AC_R	NA-R	AC_R	AC_R	AC_R
8	Read	AC_R	NA-R	APE-R	NA-R	AC_R	AE-R	NA-R	AC_R	AC_R	AC_R
9	Write	AC_W	NA-W	APE-W	NA-W	AC_W	AC_W	NA-W	AC_W	AC_W	AC_W
10	Read	AC_R	NA-R	APE-R	NA-R	AC_R	AC_R	NA-R	AC_R	AC_R	AC_R
11	Read	AC_R	NA-R	APE-R	NA-R	AC_R	AC_R	NA-R	AC_R	AC_R	AC_R
12	Write	AC_W	NA-W	APE-W	NA-W	AC_W	AC_W	NA-W	AC_W	AC_W	AC_W
13	Read	AC_R	NA-R	APE-R	NA-R	AC_R	AC_R	NA-R	AC_R	AC_R	AC_R
14	Write	AC_W	NA-W	APE-W	NA-W	AC_W	AC_W	NA-W	AC_W	AC_W	AC_W
15	Read	AC_R	NA-R	APE-R	NA-R	AC_R	AC_R	NA-R	AE-R	AC_R	AC_R
16	Read	AC_R	NA-R	APE-R	NA-R	AC_R	AC_R	NA-R	AC_R	AE-R	AC_R
17	Read	AC_R	NA-R	APE-R	NA-R	AC_R	AC_R	NA-R	AC_R	AC_R	AE-R
18	Read	AC_R	NA-R	APE-R	NA-R	AC_R	AC_R	NA-R	AC_R	AC_R	AC_R
19	Read	AC_R	NA-R	APE-R	NA-R	AC_R	AC_R	NA-R	AC_R	AC_R	AC_R
20	Read	AC_R	NA-R	APE-R	NA-R	AC_R	AC_R	NA-R	AC_R	AC_R	AC_R
21	Read	AC_R	NA-R	APE-R	NA-R	AC_R	AC_R	NA-R	AC_R	AC_R	AC_R
22	Write	AC_W	NA-W	APE-W	NA-W	AC_W	AC_W	NA-W	AC_W	AC_W	AC_W
2475	Write	AC_W	NA-W	APE-W	NA-W	AC_W	AC_W	NA-W	AC_W	AC_W	AC_W
2476	Write	AC_W	NA-W	APE-W	NA-W	AC_W	AC_W	NA-W	AC_W	AC_W	AC_W
2477	Write	AC_W	NA-W	APE-W	NA-W	AC_W	AC_W	NA-W	AC_W	AC_W	AC_W
2478	Read	AC_R	NA-R	APE-R	NA-R	AC_R	AC_R	NA-R	AC_R	AC_R	AC_R
2479	Read	AC_R	NA-R	APE-R	NA-R	AC_R	AC_R	NA-R	AC_R	AC_R	AC_R
2480	Write	AC_W	NA-W	APE-W	NA-W	AC_W	AC_W	NA-W	AC_W	AC_W	AC_W
2481	Write	AC_W	NA-W	APE-W	NA-W	AC_W	AC_W	NA-W	AC_W	AC_W	AC_W
2482	Read	AC_R	NA-R	APE-R	NA-R	AC_R	AC_R	NA-R	AC_R	AC_R	AC_R
2483	Read	AC_R	NA-R	APE-R	NA-R	AC_R	AC_R	NA-R	AC_R	AC_R	AC_R
2484	Read	AC_R	NA-R	APE-R	NA-R	AC_R	AC_R	NA-R	AC_R	AC_R	AC_R
2485	Write	AC_W	NA-W	APE-W	NA-W	AC_W	AC_W	NA-W	AC_W	AC_W	AC_W
2486	Write	AC_W	NA-W	APE-W	NA-W	AC_W	AC_W	NA-W	AC_W	AC_W	AC_W
2487	Write	AC_W	NA-W	APE-W	NA-W	AC_W	AC_W	NA-W	AC_W	AC_W	AC_W
2488	Read	AC_R	NA-R	APE-R	NA-R	AC_R	AC_R	NA-R	AC_R	AC_R	AC_R
2489	Read	AC_R	NA-R	APE-R	NA-R	AC_R	AC_R	NA-R	AC_R	AC_R	AC_R
2490	Write	AC_W	NA-W	APE-W	NA-W	AC_W	AC_W	NA-W	AC_W	AC_W	AC_W
2491	Write	AC_W	NA-W	APE-W	NA-W	AC_W	AC_W	NA-W	AC_W	AC_W	AC_W
2492	Read	AC_R	NA-R	APE-R	NA-R	AC_R	AC_R	NA-R	AC_R	AC_R	AC_R
2493	Read	AC_R	NA-R	APE-R	NA-R	AC_R	AC_R	NA-R	AC_R	AC_R	AC_R
2494	Read	AC_R	NA-R	APE-R	NA-R	AC_R	AC_R	NA-R	AC_R	AC_R	AC_R
2495	Write	AC_W	NA-W	APE-W	NA-W	AC_W	AC_W	NA-W	AC_W	AC_W	AC_W
2496	Write	AC_W	NA-W	APE-W	NA-W	AC_W	AC_W	NA-W	AC_W	AC_W	AC_W
2497	Write	AC_W	NA-W	APE-W	NA-W	AC_W	AC_W	NA-W	AC_W	AC_W	AC_W
2498	Read	AC_R	NA-R	APE-R	NA-R	AC_R	AC_R	NA-R	AC_R	AC_R	AC_R
2499	Read	AC_R	NA-R	APE-R	NA-R	AC_R	AC_R	NA-R	AC_R	AC_R	AC_R
2500	Write	AC_W	NA-W	APE-W	NA-W	AC_W	AC_W	NA-W	AC_W	AC_W	AC_W

Figure 4. Status of the nodes for case 1-A

Case 2-B: No. of. Runs =4000, workload = Read dominated, RS (n, k) = (14, 10), % of Hybrid failures = 50% and the status of nodes for case 2-B can be referred in Figure 5.

Case 3-C: No. of. Runs = 5500, workload = Write dominated, RS (n, k) = (6, 3), % of Hybrid failures = 75% and the status of nodes for case 3-C can be referred in Figure 6.

Varying cases of errors and erasures were injected in the DSS. It is observed that despite adversarial effect and failures varying from (25% to 75%), files were reconstructed successfully. RS code reconstructs the files successfully as long as the combination of errors and erasures are less than or equal to (*n-k*) / 2 and *n-k* respectively. It is observed that, the file reconstruction fails if it exceeds this factor. Once a node is crashed it remains the same at every run because it requires a repair mechanism to correct that node. Similarly the static adversary will also remain at same node throughout its execution. Mobile adversary will travel only to its neighbor node in the execution which is the nature of mobile adversary as illustrated in the test cases.

case 2-B	runs=4000		Read dominated			% =50					
Run#	Workload	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
1	Write	NA-W	NA-W	AE-W	APE-W	AC_W	AC_W	AC_W	AE-W	AC_W	AC_W
2	Write	NA-W	NA-W	AC_W	APE-W	AC_W	AC_W	AC_W	AC_W	AC_W	AC_W
3	Read	NA-R	NA-R	AC_R	APE-R	**AE-R**	AC_R	AC_R	AC_R	AC_R	AC_R
4	Write	NA-W	NA-W	AC_W	APE-W	AC_W	**AE-W**	AC_W	AC_W	AC_W	AC_W
5	Read	NA-R	NA-R	AC_R	APE-R	AC_R	AC_R	**AE-R**	AC_R	AC_R	AC_R
6	Read	NA-R	NA-R	AC_R	APE-R	AC_R	AC_R	AC_R	AC_R	AC_R	AC_R
7	Read	NA-R	NA-R	AC_R	APE-R	AC_R	AC_R	AC_R	AC_R	**AE-R**	AC_R
8	Read	NA-R	NA-R	AC_R	APE-R	AC_R	AC_R	AC_R	AC_R	AC_R	**AE-R**
9	Write	NA-W	NA-W	AC_W	APE-W	AC_W	AC_W	AC_W	AC_W	AC_W	AC_W
10	Read	NA-R	NA-R	AC_R	APE-R	AC_R	AC_R	AC_R	AC_R	AC_R	AC_R
11	Read	NA-R	NA-R	AC_R	APE-R	AC_R	AC_R	AC_R	AC_R	AC_R	AC_R
12	Write	NA-W	NA-W	AC_W	APE-W	AC_W	AC_W	AC_W	AC_W	AC_W	AC_W
13	Read	NA-R	NA-R	AC_R	APE-R	AC_R	AC_R	AC_R	AC_R	AC_R	AC_R
14	Write	NA-W	NA-W	AC_W	APE-W	AC_W	AC_W	AC_W	AC_W	AC_W	AC_W
15	Read	NA-R	NA-R	AC_R	APE-R	AC_R	AC_R	AC_R	AC_R	AC_R	AC_R
16	Read	NA-R	NA-R	AC_R	APE-R	AC_R	AC_R	AC_R	AC_R	AC_R	AC_R
17	Read	NA-R	NA-R	AC_R	APE-R	AC_R	AC_R	AC_R	AC_R	AC_R	AC_R
18	Read	NA-R	NA-R	AC_R	APE-R	AC_R	AC_R	AC_R	AC_R	AC_R	AC_R
19	Read	NA-R	NA-R	AC_R	APE-R	AC_R	AC_R	AC_R	AC_R	AC_R	AC_R
20	Read	NA-R	NA-R	AC_R	APE-R	AC_R	AC_R	AC_R	AC_R	AC_R	AC_R
21	Read	NA-R	NA-R	AC_R	APE-R	AC_R	AC_R	AC_R	AC_R	AC_R	AC_R
22	Write	NA-W	NA-W	AC_W	APE-W	AC_W	AC_W	AC_W	AC_W	AC_W	AC_W
1181	Read	NA-R	NA-R	AC_R	APE-R	AC_R	AC_R	AC_R	APE-R	AC_R	AC_R
1182	Read	NA-R	NA-R	AC_R	APE-R	AC_R	AC_R	AC_R	APE-R	AC_R	AC_R
1183	Read	NA-R	NA-R	AC_R	APE-R	AC_R	AC_R	AC_R	APE-R	AC_R	AC_R
1184	Read	NA-R	NA-R	AC_R	APE-R	AC_R	AC_R	AC_R	APE-R	AC_R	AC_R
1185	Read	NA-R	NA-R	AC_R	APE-R	AC_R	AC_R	AC_R	APE-R	AC_R	AC_R
1186	Read	NA-R	NA-R	AC_R	APE-R	AC_R	AC_R	AC_R	APE-R	AC_R	AC_R
1187	Read	NA-R	NA-R	AC_R	APE-R	AC_R	AC_R	AC_R	APE-R	AC_R	AC_R
1188	Read	NA-R	NA-R	AC_R	APE-R	AC_R	AC_R	AC_R	APE-R	AC_R	AC_R
1189	Read	NA-R	NA-R	AC_R	APE-R	AC_R	AC_R	AC_R	APE-R	AC_R	AC_R
1190	Write	NA-W	NA-W	AC_W	APE-W	AC_W	AC_W	AC_W	APE-W	AC_W	AC_W
1191	Write	NA-W	NA-W	AC_W	APE-W	AC_W	AC_W	AC_W	APE-W	AC_W	AC_W
1192	Write	NA-W	NA-W	AC_W	APE-W	AC_W	AC_W	AC_W	APE-W	AC_W	AC_W
1193	Read	NA-R	NA-R	AC_R	APE-R	AC_R	AC_R	AC_R	APE-R	AC_R	AC_R
1194	Write	NA-W	NA-W	AC_W	APE-W	AC_W	AC_W	AC_W	APE-W	AC_W	AC_W
1195	Write	NA-W	NA-W	AC_W	APE-W	AC_W	AC_W	AC_W	APE-W	AC_W	AC_W
1196	Write	NA-W	NA-W	AC_W	APE-W	**AE-W**	AC_W	AC_W	APE-W	AC_W	AC_W
1197	Read	NA-R	NA-R	AC_R	APE-R	AC_R	**AE-R**	AC_R	APE-R	AC_R	AC_R
1198	Read	NA-R	NA-R	AC_R	APE-R	AC_R	AC_R	**AE-R**	APE-R	AC_R	AC_R
1199	Read	NA-R	NA-R	AC_R	APE-R	AC_R	AC_R	AC_R	APE-R	AC_R	AC_R
1200	Read	NA-R	NA-R	AC_R	APE-R	AC_R	AC_R	AC_R	APE-R	AC_R	AC_R
1201	Read	NA-R	NA-R	AC_R	APE-R	AC_R	AC_R	AC_R	APE-R	AC_R	AC_R
1202	Read	NA-R	NA-R	AC_R	APE-R	AC_R	AC_R	AC_R	APE-R	AC_R	AC_R
1203	Read	NA-R	NA-R	AC_R	APE-R	AC_R	AC_R	AC_R	APE-R	AC_R	AC_R
1204	Read	NA-R	NA-R	AC_R	APE-R	AC_R	AC_R	AC_R	APE-R	AC_R	AC_R
1205	Read	NA-R	NA-R	AC_R	APE-R	AC_R	AC_R	AC_R	APE-R	AC_R	AC_R
1206	Write	NA-W	NA-W	AC_W	APE-W	AC_W	AC_W	AC_W	APE-W	AC_W	AC_W

Figure 5. Status of the nodes for case 2-B

3.4 Analysis

Based on the obtained results, the following analysis is performed. Figure 7 illustrates the Read success rate for various (*n,k*) values for both replication and dispersal technique.

Figure 7 it is clear that we need to access *k* servers to reconstruct the original data in case of Dispersal technique and security is also increased compared to that of Replication technique.

case 3-C	runs=5500	% =75	write dominated								
Run#	Workload	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
1	Write	NA-W	AC_W	AC_W	APE-W	NA-W	AC_W	NA-W	APE-W	AC_W	AC_W
2	Write	NA-W	AC_W	AC_W	APE-W	NA-W	AC_W	NA-W	APE-W	AC_W	AC_W
3	Read	NA-R	AC_R	AC_R	APE-R	NA-R	AC_R	NA-R	APE-R	AC_R	AC_R
4	Write	NA-W	AC_W	AC_W	APE-W	NA-W	AC_W	NA-W	APE-W	AC_W	AC_W
5	Read	NA-R	AC_R	AC_R	APE-R	NA-R	AC_R	NA-R	APE-R	AC_R	AC_R
6	Read	NA-R	AC_R	AC_R	APE-R	NA-R	AC_R	NA-R	APE-R	AC_R	AC_R
7	Read	NA-R	AC_R	AC_R	APE-R	NA-R	AC_R	NA-R	APE-R	AC_R	AC_R
8	Read	NA-R	AC_R	AC_R	APE-R	NA-R	AC_R	NA-R	APE-R	AC_R	AC_R
9	Write	NA-W	AC_W	AC_W	APE-W	NA-W	AC_W	NA-W	APE-W	AC_W	AC_W
10	Read	NA-R	AC_R	AC_R	APE-R	NA-R	AC_R	NA-R	APE-R	AC_R	AC_R
11	Write	NA-W	AC_W	AC_W	APE-W	NA-W	AC_W	NA-W	APE-W	AC_W	AC_W
12	Write	NA-W	AC_W	AC_W	APE-W	NA-W	AC_W	NA-W	APE-W	AC_W	AC_W
13	Read	NA-R	AC_R	AC_R	APE-R	NA-R	AC_R	NA-R	APE-R	AC_R	AC_R
14	Write	NA-W	AC_W	AC_W	APE-W	NA-W	AC_W	NA-W	APE-W	AC_W	AC_W
15	Write	NA-W	AC_W	AC_W	APE-W	NA-W	AC_W	NA-W	APE-W	AC_W	AC_W
16	Read	NA-R	AC_R	AC_R	APE-R	NA-R	AC_R	NA-R	APE-R	AC_R	AC_R
17	Write	NA-W	AC_W	AC_W	APE-W	NA-W	**AE-W**	NA-W	APE-W	AC_W	AC_W
18	Read	NA-R	AC_R	AC_R	APE-R	NA-R	AC_R	NA-R	APE-R	AC_R	AC_R
19	Write	NA-W	AC_W	AC_W	APE-W	NA-W	AC_W	NA-W	APE-W	AC_W	AC_W
20	Read	NA-R	AC_R	AC_R	APE-R	NA-R	AC_R	NA-R	APE-R	**AE-R**	AC_R
21	Write	NA-W	**AE-W**	AC_W	APE-W	NA-W	AC_W	NA-W	APE-W	AC_W	**AE-W**
22	Write	NA-W	AC_W	**AE-W**	APE-W	NA-W	AC_W	NA-W	APE-W	AC_W	AC_W
5475	Write	NA-W	AC_W	AC_W	APE-W	NA-W	AC_W	NA-W	APE-W	AC_W	AC_W
5476	Write	NA-W	AC_W	AC_W	APE-W	NA-W	AC_W	NA-W	APE-W	AC_W	AC_W
5477	Write	NA-W	AC_W	AC_W	APE-W	NA-W	AC_W	NA-W	APE-W	AC_W	AC_W
5478	Write	NA-W	AC_W	AC_W	APE-W	NA-W	AC_W	NA-W	APE-W	AC_W	AC_W
5479	Read	NA-R	AC_R	AC_R	APE-R	NA-R	AC_R	NA-R	APE-R	AC_R	AC_R
5480	Read	NA-R	AC_R	AC_R	APE-R	NA-R	AC_R	NA-R	APE-R	AC_R	AC_R
5481	Write	NA-W	AC_W	AC_W	APE-W	NA-W	AC_W	NA-W	APE-W	AC_W	AC_W
5482	Write	NA-W	AC_W	AC_W	APE-W	NA-W	AC_W	NA-W	APE-W	AC_W	AC_W
5483	Write	NA-W	AC_W	AC_W	APE-W	NA-W	AC_W	NA-W	APE-W	AC_W	AC_W
5484	Read	NA-R	AC_R	AC_R	APE-R	NA-R	AC_R	NA-R	APE-R	AC_R	AC_R
5485	Write	NA-W	AC_W	AC_W	APE-W	NA-W	AC_W	NA-W	APE-W	AC_W	AC_W
5486	Write	NA-W	AC_W	AC_W	APE-W	NA-W	AC_W	NA-W	APE-W	AC_W	AC_W
5487	Write	NA-W	AC_W	AC_W	APE-W	NA-W	AC_W	NA-W	APE-W	AC_W	AC_W
5488	Write	NA-W	AC_W	AC_W	APE-W	NA-W	AC_W	NA-W	APE-W	AC_W	AC_W
5489	Write	NA-W	AC_W	AC_W	APE-W	NA-W	AC_W	NA-W	APE-W	AC_W	AC_W
5490	Read	NA-R	AC_R	AC_R	APE-R	NA-R	AC_R	NA-R	APE-R	AC_R	AC_R
5491	Read	NA-R	**AE-R**	AC_R	APE-R	NA-R	AC_R	NA-R	APE-R	AC_R	AC_R
5492	Write	NA-W	AC_W	**AE-W**	APE-W	NA-W	AC_W	NA-W	APE-W	AC_W	AC_W
5493	Write	NA-W	AC_W	AC_W	APE-W	NA-W	AC_W	NA-W	APE-W	AC_W	AC_W
5494	Write	NA-W	AC_W	AC_W	APE-W	NA-W	AC_W	NA-W	APE-W	AC_W	AC_W
5495	Read	NA-R	AC_R	AC_R	APE-R	NA-R	AC_R	NA-R	APE-R	AC_R	AC_R
5496	Write	NA-W	AC_W	AC_W	APE-W	NA-W	AC_W	NA-W	APE-W	AC_W	AC_W
5497	Write	NA-W	AC_W	AC_W	APE-W	NA-W	AC_W	NA-W	APE-W	AC_W	AC_W
5498	Write	NA-W	AC_W	AC_W	APE-W	NA-W	AC_W	NA-W	APE-W	AC_W	AC_W
5499	Write	NA-W	AC_W	AC_W	APE-W	NA-W	AC_W	NA-W	APE-W	AC_W	AC_W
5500	Write	NA-W	AC_W	AC_W	APE-W	NA-W	AC_W	NA-W	APE-W	AC_W	AC_W

Figure 6. Status of the nodes for case 3-C

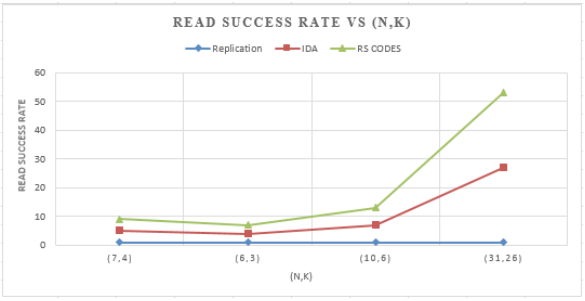

Figure 7. Read Success Rate

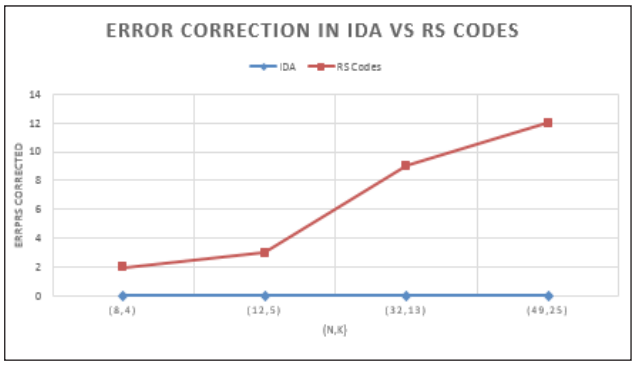

Figure 8. Error Tolerance in IDA and RS codes

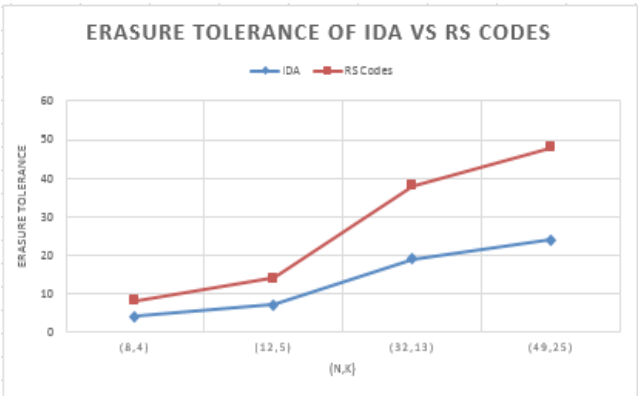

Figure 9. Erasure Tolerance in IDA and RS codes

Figure 8 illustrates that RS codes outperforms IDA in terms of Error correcting capability. Both are equivalent in Erasure tolerance as shown in Figure 9.

4. Conclusion and Future Enhancement

Research in DSS using Quorum based approaches have strived to provide availability, consistency and fault tolerance. They are basically oriented to the replication approach and lack in security (Confidentiality). A classic and secure alternative to replication is dispersal. Dispersal using Reed Solomon codes stand out in their ability to handle both Crash and Non-crash failures at DSS as brought out in section 3. Dual Quorum approach with dispersal DQ-D, brings out the ability of the design for availability, consistency, fault tolerance and security. But does not consider Hybrid failures. This extension is considered in the present work to analyze the performance of DQ-Dispersal with DQ-Replication in the presence of hybrid failures is in progress.

5. Acknowledgment

I would like to thank my Guide Dr. C.K. Shyamala, Assistant Professor, Department of Computer Science and Engineering, Amrita Vishwa Vidyapeetham, Coimbatore for her guidance, reviews and valuable comments which helped a lot in writing this paper.

6. References

1. Bansal S, Sharma S, Trivedi I, A detailed review of fault-tolerance techniques in Distributed System. *International Journal on Internet and Distributed Computing Systems*. 2011 Jun; 1(1):33.
2. Zhao D, Burlingame K, Debains C, Alvarez-Tabio P, Raicu I. Towards high-performance and cost-effective distributed storage systems with information dispersal algorithms. In: *Cluster Computing (CLUSTER), IEEE International Conference*; 2013 Sep. p. 1–5 .
3. Bal HE, Kaashoek MF, Tanenbaum AS, Jansen J. Replication techniques for speeding up parallel applications on distributed systems. *Concurrency: Practice and Experience*. 1992 Aug; 4(5):337–355.
4. Nirmala SJ, Bhanu SM, Patel AA. A Comparative study of the secret sharing algorithms for secure data in the cloud. *International Journal on Cloud Computing: Services and Architecture (IJCCSA)*. 2012 Aug; 2(4):63–71.
5. Abdallah A, Salleh M. Analysis and Comparison the Security and Performance of Secret Sharing Schemes. *Asian Journal of Information Technology*. 2015; 14(2):74–83.
6. Kumar S, Gupta R. Bit error rate analysis of Reed-Solomon code for efficient communication system. *International Journal of Computer Applications*. 2011 Sep; 30(12):11–15.
7. Shyamala CK, Vidya NV. Structuring Reliable Distributed Storages. In Intelligent Systems Technologies and Applications, Springer International Publishing. 2016; 235–245.
8. Bose R. Information theory, coding and cryptography. Tata McGraw-Hill Education; 2008.
9. Rabin MO. Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the ACM (JACM)*. 1989 Apr; 36(2):335–48.
10. Ling J, Jiang X. Distributed storage method based on information dispersal algorithm. *In Instrumentation and Measurement, Sensor Network and Automation (IMSNA), 2013 2nd International Symposium;* 2013 Dec. p. 624–626.
11. Debains C, Alvarez-Tabio P, Zhao D, Raicu I. IStore: Towards High Efficiency, Performance, and Reliability in Distributed Data Storage with Information Dispersal Algorithms. under review at IEEE MSST; 2013 Jan. p. 1–8.
12. Peleg D, Wool A. The availability of quorum systems. *Information and Computation*. 1995 Dec; 123(2):210–223.

13. Gao L, Dahlin M, Zheng J, Alvisi L, Iyengar A. Dual-Quorum: A Highly Available and Consistent Replication System for Edge Services. *Dependable and Secure Computing, IEEE Transactions*. 2010 Apr; 7(2):159–174.
14. Bowers KD, Juels A, Oprea A. HAIL: a high-availability and integrity layer for cloud storage. In: *Proceedings of the 16th ACM Conference on Computer and Communications Security;* 2009 Nov. p. 187–198 .
15. Shyamala CK, Vidya NV. Dual Quorum-Dispersal, A Novel Approach for Reliable and Consistent ECC based storages. *International Journal of Applied Engineering and Research (IJAER)*.2015; 10(3):6901–6917.
16. Martin KM. Challenging the adversary model in secret sharing schemes. Coding and Cryptography II, *Proceedings of the Royal Flemish Academy of Belgium for Science and the Arts;* 2008. p. 45–63.
17. Shyamala CK, Sharanya RR. Study of Adversary Models in Distributed Storage Systems. *Global Journal of Pure and Applied Mathematics (GJPAM)*. 2015; 11(1): 148–155.