

**SIMULATION OF STARFISH:
STABLE AND TAMPER
RESISTANT FILE-SHARING**

PROJECT REPORT

p-1195

*Submitted in partial fulfillment of the
Requirement for the award of the degree of*

Bachelor of Engineering

In

Computer Science

Of

Bharathiar University, Coimbatore.

Submitted by

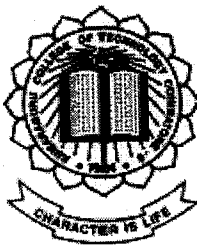
Vijayalakshmi. J

0027KO210

Under the expert Guidance of

Mrs. J. Cynthia, M.E.,

Lecturer,



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING,
KUMARAGURU COLLEGE OF TECHNOLOGY,
COIMBATORE – 641 006.**

MARCH 2004.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
KUMARAGURU COLLEGE OF TECHNOLOGY
(Affiliated to Bharathiar University, Coimbatore)

CERTIFICATE

This is to certify that the project entitled

**SIMULATION OF STARFISH:
STABLE AND TAMPER
RESISTANT FILE-SHARING**

has been done by

Vijayalakshmi. J
0027KO210

*and submitted in partial fulfillment of the Requirement
for the award of the degree of*

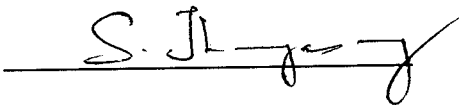
Bachelor of Engineering

In

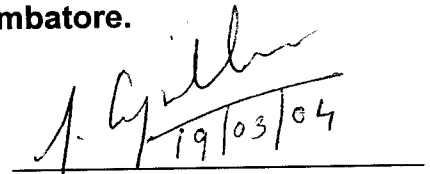
Computer Science

Of

Bharathiar University, Coimbatore.



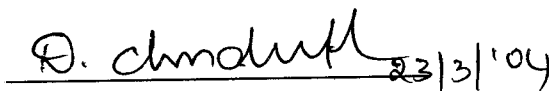
Professor & Head of the department
(Prof. S. THANGASAMY)



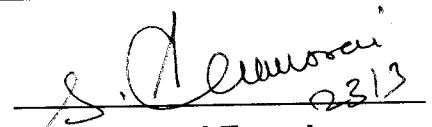
Guide
(Mrs. J. Cynthia)

Certified that the candidate was examined by us in the project work

Viva voce examination held on 23.3.04



Internal Examiner



External Examiner

Declaration

DECLARATION

I,

Vijayalakshmi. J 0027K0210

declare that the project entitled "**SIMULATION OF STARFISH: STABLE AND TAMPER RESISTANT FILE-SHARING**", has been done by me and to the best of my knowledge, a similar work has not been submitted to the Bharathiar University or any other institution, for fulfillment of the requirements of the course study.

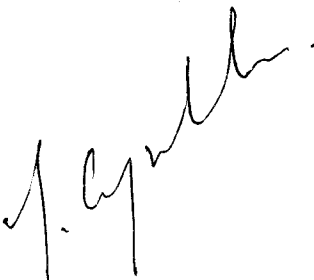
This report is submitted on the partial fulfillment of the requirements for all awards of degree of Bachelor of Computer Science and Engineering of Bharathiar University.

Place: Coimbatore

J. Vijayalakshmi

J. Vijayalakshmi

Date: 22.3.04



[Guided by: Mrs. J. Cynthia, M.E.,]

Acknowledgments

ACKNOWLEDGMENTS

The exhilaration achieved upon the successful completion of any task should be definitely shared with the people behind the venture. This project is an amalgam of study and experience of many people without whose help this project would not have taken shape.

At the outset, I take this opportunity to thank the management of my college for having provided us excellent facilities to work with. I also wish to express my deep gratitude to our Principal **Dr. K. K. Padmanabhan**, B.Sc.,(Engg), M.Tech., Ph.D., for ushering us in the path to triumph.

I am always thankful to our beloved Professor and the Head of the Department, **Prof. S. Thangasamy** B.E.(HONS)., whose consistent support and enthusiastic involvement helped us a great deal.

I am greatly indebted to my beloved guide **Mrs. J. Cynthia**, M.E., Lecturer, Department of Computer Science and Engineering for her excellent guidance and timely support during the course of this project. As a token of my esteem and gratitude, I honor her for her assistance towards this cause.

I also thank my project coordinator **Ms. S. Chandrakala** M.E., and my beloved class advisor **Ms. Hema Gupta** B.E., for their invaluable assistance.

I also feel elated in manifesting my deep sense of gratitude to all the staff and lab technicians of the Department of Computer Science and Engineering.

I am profoundly grateful to my Parents for their enthusiasm, encouragement, loving support, and motivation. I especially thank my elder brother without whose invaluable support and guidance at critical junctures, this project would not have been a success. I also thank my younger brother for his constructive criticism.

Lastly, I thank all my friends, for contributing useful comments and technical information as and when needed.

Synopsis

SYNOPSIS

Recently, work on Peer-to-Peer file sharing systems has multiplied, both in industry [1-3] and in academia [4-6]. This work has fundamentally altered the way in which popular data, especially multimedia data, has been accessed on the Internet. Along with the initial applications of file-sharing, there have also been new applications in file storage [4,5], which promise an always-on, global scale file storage facility using a potentially unlimited amount of storage. I propose a new file-sharing project called **"Simulation of STARFISH: Stable and TAMper Resistant File SHaring"**. STARFISH uses techniques from erasure coding theory to construct a file-sharing system with the following desirable properties:

RELIABILITY:

Nodes in P2p systems cannot be relied on to be always connected to the Internet, or even if they do, to execute data access requests correctly. I propose to use erasure correcting codes, or in general error-correcting codes to build a reliable data sharing system that can function correctly even when a subset of providers crash or misbehave.

CONCURRENCY:

Reliable storage systems should permit concurrent access by multiple clients efficiently. Also fully de-centralized systems build over P2P systems should do so without centralized mechanisms like lock servers etc. I propose the use of concurrent write primitives in building a highly efficient file sharing system that can handle concurrent accesses in a fully distributed manner.

For this, we can use a simple parity code scheme. There are two data disks and one parity disk. The Client module has two commands: read and write and we start up the client to read from and write to disks. Clients write blocks to data disks and update the parity. If a disk fails, it can be recovered using the other data disk and the parity disk.

Contents

CONTENTS

1. Introduction	1
1.1 Existing system and its limitations	1
1.2 Proposed system and its advantages	2
2. Literature Survey	4
3. System Requirements Analysis	6
3.1 Product definition	6
3.2 Project Plan	7
4. Software Requirements Specification	8
4.1 Purpose	8
4.2 Scope	8
4.3 Product overview and summary	8
4.4 Development and operating environment	9
4.5 External interfaces and data flow specifications	9
4.6 Functional Specifications	10
4.6 Exception handling	12
4.7 Product Optimization	12
5. System Design	13
5.1 Input Design	13
5.2 Output Design	13
5.3 Process Design	13
6. System testing	18
7.1 Testing objectives	18
7.2 Levels of testing	18

7. Future Enhancements	20
8. Conclusion	21
9. References	22
10. Appendices	23
10.1 Sample source code	23
10.2 Sample output	40

Introduction

1. INTRODUCTION

Distributed file-sharing applications, like the Gnutella [1] network, are starting to revolutionize the way users access media on the Internet. This project **"SIMULATION OF STARFISH: Stable and Tamper Resistant File-Sharing"** proposes the simulation of the design and implementation of a scalable and fault-tolerant file-sharing system built using ideas from Error-correcting coding theory.

A (n, k) erasure correcting code takes k data blocks and encodes it into n encoded blocks, out of which any k encoded blocks suffice to recover the original data blocks. Thus, these codes can be used to implement storage applications that can recover from the loss of some of its storage units. The flexibility of such an approach also promises higher performance by allowing clients to load-balance requests between multiple storage units. Through analysis and experimentation, I aim to demonstrate the advantages of the erasure-code-based approach over traditional file-sharing techniques.

1.1 Existing System and its Limitations

Large enterprise storage systems increasingly employ disks that attach to a network rather than to a single node. This means the disks can be accessed by multiple nodes in the network, which provides added flexibility and fault tolerance. Erasure codes provide an efficient way to encode data into multiple fragments, such that if some fragments are lost, the original data can still be recovered. If each fragment is stored in a separate disk, then a limited number of disk failures will not result in loss of information.

When multiple nodes can access the disks simultaneously, a problem arises: concurrent updates to the erasure code may result in inconsistencies. For example, suppose that data a and b are stored on four sectors on different

disks, each containing a , b , $a+b$ and $a-b$ respectively, represented as $(a, b, a+b, a-b)$. This is an erasure code where any two of the four sectors can be used to reconstruct a and b . Now suppose that a node p_1 wants to change a to c while a node p_2 wants to change b to d . If p_1 and p_2 concurrently write the new sectors, then the interleaving of writes may leave the system in an inconsistent state, like $(c, d, c+b, a-d)$.

Previous solutions to this problem, are based on locks or two-phase commit protocols. These solutions are quite inefficient in terms of time and messages: with locks, nodes need to acquire and release a lock for each update, whereas with two-phase commit protocols, nodes need to contact disks twice for each update. In a shared memory system, the broad idea of implementing objects that do not fail from objects that may fail has been an interesting area of study, analysis and research.

1.2 Proposed system and its advantages

The goal is to provide a solution that avoids these expensive synchronization mechanisms most of the time. Furthermore, we want to tolerate node failures, that we can implement with cheap commodity parts, rather than expensive and complex fault-tolerant hardware.

Finally, we want a comprehensive solution that not just tolerates failures, but also allows online recovery of nodes and disks. The proposed scheme is quite different because the Clients share all disks, the Clients do not talk to each other (they only talk to the shared disk objects), and they do not require non-volatile memory or synchronized clocks. The work in [3] proposes to use locks or two-phase commit to coordinate every concurrent write. The proposed scheme avoids using locks or two-phase commit in the common case.

Literature Survey

2. LITERATURE SURVEY

The Literature survey for this project has been done in the area of Erasure correcting codes in general and on X-codes in detail. Erasure codes have been in existence for decades and they have been studied in detail for their possible uses in the areas of distributed storage and communication.

Recall how erasure code based schemes work: A (n, k) erasure correcting code can take k data blocks and encode it into n encoded blocks, out of which any k encoded blocks suffice to recover the original data blocks. Thus, these codes can be used to implement storage applications that can recover from the loss of some of its storage units. This project has been inspired by a few tutorials and papers on the above-mentioned areas.

[L. Xu et al. 1999] discusses the new class of MDS code, the X-Code. The X-codes are of *minimum column distance* 3, namely they can correct either one *column error* or two *column erasures*. The key novelty in X-code is that it has a simple geometrical construction which achieves encoding/update optimal complexity i.e., A change in any single information bit affects *exactly two* parity bits. The key idea is that all parity symbols are placed in rows rather than columns.

A common property of this code is that the encoding and decoding procedures use only simple XOR and cyclic shift operations, and thus are more efficient than Reed-Solomon codes in terms of computational complexity. The two parity rows are calculated by computing the parities along the two diagonals of slope +1 and -1. In an array code-based fault-tolerant storage scheme, since each column represents a disk, this also means that the parity blocks are distributed evenly over all the disks, resulting in balanced disk access during parity updates.

System

Requirements

Analysis

3. System Requirement Analysis

System study is an activity that encompasses most of the tasks that we have collectively called computer system engineering. System study is conducted with the following objectives:

- Identify the needs.
- Evaluate the system concept for feasibility.
- Perform economic and technical analysis.
- Allocate function to hardware, software, people and other system elements.
- Create a system definition that forms the foundation for all subsequent engineering works.

3.1 Product definition

To summarize, the contributions of the project are the following:

- The design of a loosely coupled distributed file sharing system based on erasure codes, where disks are supplemented with very simple functionality like swap and add operations.
- An algorithm for writing and reading data that maintains consistency.
- A simple and efficient code that the Disk executes when it receives a swap or a Δ -Write request from the client.
- Simulation of the design and implementation of STARFISH, done using the Java™ Programming language.
- Generalizing regular registers as the correctness condition of the algorithm.
- Proving correctness of the algorithm and analyzing its resiliency to failures.

Software

Requirements

Specification

4. SOFTWARE REQUIREMENTS SPECIFICATION

4.1 Purpose:

The purpose of this project is to design a distributed and fault tolerant file sharing system based on Erasure correcting codes. The primary concerns are that the developed system is reliable and supports concurrent access. To illustrate the same, a simulation program in Java is also desired.

4.2 Scope:

The key attribute in the scope of the developed system is the level of fault tolerance. As such, this system supports concurrent access by multiple clients and can withstand one disk failure. When more than one disks fail, the data cannot be reconstructed. This can be circumvented by using a (5,3) code because the model and the erasure code used are scalable.

4.3 Product overview and summary:

My product provides a reliable, robust and efficient means of distributed file-sharing. The project basically involves the usage of erasure correcting codes for reliability and concurrency in distributed file-sharing systems.

I use a simple parity code scheme that can withstand the loss of one storage unit, here, a disk. The client reads and writes to the data and parity disks and the manager is used for initialization, nuking and recovery of disks. When a disk fails, it can be recovered using the other data disk and the parity disk. The disks are Daemons that sit around and service the client's read and write requests, while the Client is a standalone program. The client communicates its requests and receives the responses through the Communication module.

4.4 Development and Operating Environment:

The development environment gives the minimum hardware and software requirements that come with most PCs, by default.

4.4.1 Hardware Specification:

Processor	Pentium III
RAM	128 MB
Hard Disk	8 GB
Floppy Drive	1.44 MB
Monitor	14" Monitor

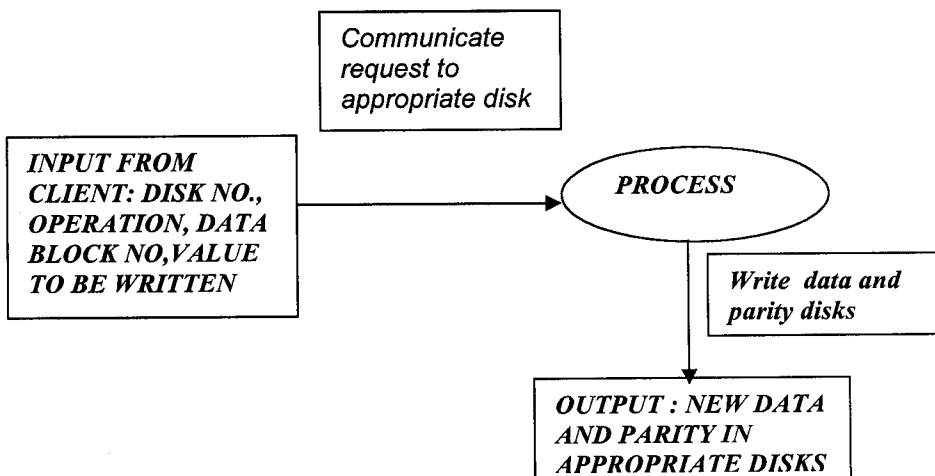
4.4.2 Software Specification:

Operating System	Windows 98
Programming Language	Java on multiple platforms

4.5 External interfaces and Dataflow:

The input for the data and parity disks is given as command line arguments at the DOS prompt or through the GUI provided. The output for the System is the data and parity disk files that are created automatically.

Data flow diagram for writing to a disk



4.6 Functional specifications:

The description of the modules is as follows:

- Disk module
- Client Module
- Erasure Code Module
- Manager Module
- The communication Module
- GUI Module

The Client is a standalone program that is called for each operation, whereas the Disk is a Daemon that sits around and services requests. The manager is included in the Client program itself. The Communication between the Client and the Disks is managed by the Communication module and the operations such as XOR are performed by the Erasure Code module.

4.6.1 Disk Module:

The Disk is a daemon that will sit around and service requests. We can use a simple parity code with two data disks and one parity disk. Clients write blocks to data disks and update the parity. When any one disk crashes, the parity is used to recover the data. The write command will use swap and add to update the disk. It will also respond to requests for swap and add from clients. After a disk is nuked, we should call the manager again with the recover command. Then the disk will start servicing requests again.

4.6.2 Client Module:

There is a client process which has two commands: read and write to read from and write to disks. Only if the Disk Daemons are started, can the client communicate with the disks. The client requests are communicated to the disk and the disk responds appropriately. Once a disk is nuked, Clients will not be able to read/write to them until the recover command is used.

4.6.3 Erasure Code Module:

It is responsible for operations on blocks like addition, subtraction etc., It will do just an XOR for parity, but for other codes like Reed-Solomon, it will be more complex.

4.6.4 Manager Module:

The manager has three commands:

- `init`
- `nuke`
- `recover`.

init initializes a disk with empty blocks.

nuke "nukes" a disk, i.e. it simulates a condition where the disk crashes and won't respond to requests

recover recovers a disk from the redundant data.

The manager will initiate a recovery when a disk crashes. When the manager finishes recovery, the data on disk will be restored.

4.6.5 The Communication Module:

This module, as already said, is used for initiating the communication between the client and the disks. The Client requests are sent to the disk and processed and the appropriate action is taken depending on the request. The response is given to the client indicating the success or failure of the operation, as the case may be.

4.6.6 The GUI Module:

In this module, an appropriate Graphical User Interface(GUI) is provided which makes the product easy to use and operate. All the options are given in a very simple and easy-to-use format, from which the user can choose.

4.7 Exception handling:

The package can handle only text. If any other type of input is used a warning message will be displayed. The usage of each module is also displayed when the user fails to provide some of the arguments.

4.8 Product Optimization:

The product can be optimized by providing adequate security mechanisms, authentication and authorization. The Product can also be optimized by using the latest version of all hardware and software components in this project.

It is easily seen that the above mentioned system can be generalized to use error-correcting codes, like the popular Reed-Solomon codes. An (n, k, d) error-correcting code takes k data symbols and constructs n encoded symbols with minimum distance of d . Such a code with minimum distance d is capable of recovering from up to $(d-1)/2$ erroneous symbols. Such codes may be used to extend STARFISH to the case of untrustworthy providers who intend to maliciously corrupt file system data. A variant of STARFISH implemented using a (n, k, d) error-correcting code can recover from up to $(d-1)/2$ malicious nodes.

System Design

5. System Design

5.1 INPUT DESIGN:

Reliable input design ensures accurate output from the system. The system to be designed is configured to receive the following data:

- The program to be invoked (client or disk etc.,)
- The Operation to be done (Read, write, recover etc.,)
- The Disk that is to be operated on (Disk1, Disk2 etc.,)
- The Block number that's is to be read from or written to
- The value to be written in case of write command

Care is taken to ensure that the inputs to the system are interactive and user friendly.

5.2 OUTPUT DESIGN:

Outputs from the system are required to communicate the result to the end user and to the client. The outputs of the system are the following:

- The Request/Response dialogues between the Client and the Disk Daemons.
- The creation, and initialization to null, of disks on startup.
- The Value stored in the block of a disk in case of a read request to that block of that disk, from the client.
- The Data blocks are stored in the data disk files Disk1 and Disk2 and the Parity Blocks, stores in the Parity Disk File Disk3.
- The recovered file in case of recovery.
- Appropriate error messages to users.

5.3 PROCESS DESIGN:

In this section, we provide background on providing fault-tolerant distributed storage using a class of erasure codes called *array codes* [7].

Array codes are two-dimensional codes in which parities are calculated along lines of different slopes. The parity blocks in array codes, in contrast with encoded blocks in general MDS codes, are calculated by bitwise XOR operation on data blocks. Hence array codes have *low computational complexity*.

Array are suitable for recovery from column erasures- erasures in which all symbols from one or more columns are lost. Thus they are useful in distributed storage schemes because in practice, when a disk fails, all blocks in it are lost. Array codes can be used in fault-tolerant storage by treating each column as a disk, and each cell in a column as one block in the disk .

An (n, k) MDS array code adds $(n-k)$ parity columns to k data columns in a way that any k columns suffice to recover the original data columns. A fault-tolerant storage system implemented using an (n, k) MDS array code can recover from k simultaneous disk failures. A useful measure of the complexity of an array code is its update complexity, which is the number of parity symbols that must be updated when a data symbol is written. Array codes with optimal update complexity are clearly desirable in read/write distributed storage applications.

X-code: An Array Code with Optimal Encoding

As already illustrated, the X-code [L.Xu et al. 1999] is an $(n, n-2)$ MDS array code with optimal update complexity: updating a data symbol requires updating exactly two parity symbols. Thus the X-code can be used to implement a fault-tolerant distributed storage system that can tolerate two simultaneous disk failures. The novel feature of the X-code is that it has parity rows, not columns. The two parity rows are calculated by computing the parities along the two diagonals of slope $+1$ and -1 . In an array code-based fault-tolerant storage scheme, since each column represents a disk, this also

means that the parity blocks are distributed evenly over all the disks, resulting in balanced disk access during parity updates.

The following algorithm may be used for the writes by the client program that is consistent but fails in the presence of concurrency.

Algorithm 1 Write algorithm executed by client, which is consistent but fails in the presence of concurrency.

```

1  procedure client-write ( $i, X_{new}$ )
2     $X_{old} \leftarrow \text{invoke}(\text{disk}(i), \text{READ}, i)$ 
3     $\text{invoke}(\text{disk}(i), \text{WRITE}, i, X_{new})$ 
4    for each  $(i, j) \in \mathbf{E}$  do
5       $P_{old} \leftarrow \text{invoke}(\text{disk}(j), \text{READ}, j)$ 
6       $P_{new} \leftarrow P_{old} + W_{ij} * (X_{new} - X_{old})$ 
7       $\text{invoke}(\text{disk}(j), \text{WRITE}, j, P_{new})$ 
8  procedure invoke ( $dest, type, \dots$ )
9    send ( $type, \dots$ ) to  $dest$ 
10   wait until receive result from  $dest$ 
11   return result

```

Theorem 1 The *write* algorithm (Algorithm 1) is consistent.

Proof sketch: As the contents of a data block X_k changes from X_{old} to X_{new} , it is clear that to keep consistency a parity block X_j needs to be updated by subtracting $W_{ij} * X_{old}$ and adding $W_{ij} * X_{new}$. This is exactly what the algorithm does.

PRIMITIVES FOR SUPPORTING CONCURRENT ACCESS

In this section, I show how to implement the write procedure (Algorithm 1) in a way that consistency is maintained when multiple clients execute it concurrently on data blocks with a shared parity block. To illustrate this problem, consider a system with one parity disk where two clients A and B and perform the write algorithm (Algorithm 1) as follows: Assume A and B both write to a data block, but then write to the corresponding parity block in

the reverse order. The parity in this case clearly becomes inconsistent with the data and the system is said to fail under concurrent access.

Existing solutions use locks or two-phase commits to make the write atomic and mutually exclusive. However these mechanisms are rather expensive and the cost needs to be paid for every write, even though only a few of them may conflict with each other.

The proposal for STARFISH is based on the two new atomic write primitives introduced in : the SWAP and the Δ -WRITE which are described in Algorithm 3. Intuitively, when the disk receives a SWAP request, it writes the new data and returns the old one. And when a disk receives a Δ -WRITE request, it replaces the old data with new data added with the old data.

In both cases, it is important that these requests be executed atomically. However, this is easy to achieve since the requests are processed locally at a single disk. Given the availability of SWAP and Δ -WRITE at the disk, a client can write a block and update its parity through Algorithm 2, which works as follows: The client first uses the SWAP request to write the new data and get the old data. The disk then computes the value Δ that needs to be added to each parity block, and then uses the Δ -WRITE request.

The following is the improved write algorithm that is consistent even under concurrent access.

Algorithm 2 Improved write algorithm executed by client, which is correct even under concurrent execution.

```

1 procedure client-write ( $i, X_{new}$ )
2    $X_{old} \leftarrow \text{invoke}(\text{disk}(i), \text{SWAP}, i, X_{new})$ 
3   for each  $(i, j) \in \mathbf{E}$  do
4      $\Delta \leftarrow W_{ij} * (X_{new} - X_{old})$ 
5      $\text{invoke}(\text{disk}(j), \Delta\text{-WRITE}, j, \Delta)$ 

```

The code executed by the disk is as illustrated below:

Algorithm 3 Code executed by the disk when it receives a SWAP or Δ -WRITE request.

```

1  upon receive (SWAP,  $i$ ,  $X_{new}$ ) from client do
2     $X_{old} \leftarrow \text{read}(X_i)$ 
3    write ( $X_i$ ,  $X_{new}$ )
4    send  $X_{old}$  to client

5  upon receive ( $\Delta$ -WRITE,  $i$ ,  $\Delta$ ) from client do
6     $X_{old} \leftarrow \text{read}(X_i)$ 
7    write ( $X_i$ ,  $X_{old} + \Delta$ )
8    send OK to client

```

The benefit of these simple algorithms are that multiple clients can run it concurrently, and the resulting parity blocks will be consistent regardless of how the execution interleaves. This is accomplished without expensive synchronization mechanisms like locks and two-phase commits.

Example

Consider a storage system that implements fault-tolerance using the (5,3) X-code. For simplicity, assume all blocks are 4-bit values. Here the + and - operators are both the bitwise binary XOR operators. Let the data blocks take on the values $a=0000, b=0001, o=1110$, so that the two parities that depend on a are $a+g+m=1010$ and $a+j+n=0100$. Now let's assume that the clients A and B want to concurrently write the value 1111 and 0101 to block a , and also, without loss of generality, that A's swap executes first. Now, A's swap returns the value 0000 and writes 1111 to a , leading A to calculate g as $1111-0000=1111$. B's swap will return the value that A just wrote, i.e., 1111 and write 0101 to disk, leading B to calculate its g as $0101-1111=1010$. Now the disk will receive two Δ -WRITE requests with $\Delta=1111$ and $\Delta=1010$.

Thus the final parity values after both writes complete, are

$$a+g+m \leftarrow a+g+m+1111+1010=1111 \text{ and}$$

$$a+g+n \leftarrow a+g+n+1111+1010=0001, \text{ which are consistent with the data values } a=0101, g=0110, m=1100, j=1001, n=1101.$$

System Testing

6. SYSTEM TESTING

Testing is an important activity in the software lifecycle and is used to verify that the system being built is correct and consistent. Its performed with the intention of finding the defects and faults in the system. Testing is however not restricted to being performed after the development phase. This is to be carried out in parallel with all stages of system development, starting with requirements specification.

Testing results once gathered and evaluated, provide an indication of the software quality and reliability and serves as a basis for design modification if required. System Testing is the process of checking whether the developed system is working according to the original objectives and requirements. The system should be tested experimentally with test data so as to ensure that it works according to the requirements specification. When the system is found working, we test it with actual data and evaluate its performance.

6.1 Testing Objectives:

The testing objectives of this project are summarized as follows:

- To check if the system really does what it is expected to do.
- To check if the main requirements of the project are met.
- To test how the system handles erroneous inputs

6.2 Levels of Testing:

The Software functionality tests and the testing procedures that have been used for this project are as follows:

- Unit Testing
- Integration Testing
- Output Testing

Unit Testing:

Unit testing has been carried out to verify and uncover errors within the boundary of the smallest unit or a module. In this testing step, each module was found to be working satisfactory as per the expected output of the module. In the package development, each module is tested separately after it has been completed and checked with valid data. The 6 logical modules of the project are checked separately and subtle bugs are found and removed.

Integration Testing:

Integration Testing address the issues associated with the dual problems of verification and program construction. After the software has been integrated a set of higher-order tests are conducted. The main objective in this testing process is to take unit-tested modules and build a program structure that has been dictated by design. In this project the collective working of the project and their interconnection structures are verified. Problems in linking the modules are discovered and appropriate corrections have been made.

Output Testing:

Although each output test has a different purpose, all the work should be verified so that all system elements have been properly integrated and perform their allocated functions. All possible kinds of inputs have given to all the modules and the outputs have been verified. Errors and Exceptions are handled by the system and an appropriate output is displayed for such cases and a suitable action is taken.

Future

Enhancements

Conclusion

8. Conclusion

Fault-tolerance and concurrency two important issues in distributed storage. In this project, I proposed methods for constructing a fault-tolerant distributed storage system capable of supporting concurrent access. Using a novel and efficient erasure code , I showed how to construct distributed storage capable of recovery from any one disk failure, which can also be scaled to recover from two simultaneous disk failures.

Writes to a data block in fault-tolerant storage involves attendant writes to parity blocks, and correct operation under concurrent access requires coordination between multiple independent agents, usually through expensive concurrency control procedures like locks on every write operation.

I have taken an alternate approach by introducing two write primitives, SWAP and -WRITE ; each write operation, when implemented as a sequence of a SWAP followed by one or more -WRITES , is serializable and may be executed concurrently on the same parity group by multiple clients. Synchronization mechanisms like locks or two-phase commits are avoided most of the time. When all concurrent clients complete their writes, the parity group is guaranteed to be consistent.

References

Appendices

10. Appendices

10.1 Sample source code

Client. Java

/* This client program is a standalone program and is used to read from and write to working disks. It communicates read/write requests to the Disk program through the request program and receives responses from the disk through the response program. It also performs the managing functions such as initialization of disks, recovery of disks, etc., */

```
import java.io.*;
import java.net.*;
import java.util.*;

/*Creating the disk properties file */
public class Client {

    private Properties diskProperties;
    int disks, blocks, blocksize;

    public Client(Properties diskProperties) {
        this.diskProperties = diskProperties;
        blocksize =
Integer.parseInt(diskProperties.getProperty("blocksize"));
        disks = Integer.parseInt(diskProperties.getProperty("disks"));
        blocks = Integer.parseInt(diskProperties.getProperty("blocks"));
    }

    private Socket getSocket(String disk) throws IOException {
        return new
Socket(InetAddress.getByAddress(diskProperties.getProperty(disk+".address"))
, Integer.parseInt(diskProperties.getProperty(disk+".port")));
    }

    /*Apply XOR*/
    public void applyxor(byte[] old, byte[] delta) {
        for (int i = 0; i < blocksize; i++) {
            old[i] = (byte)(((int)old[i]^((int)delta[i]) & 0xff);
        }
    }

    /*Represent as 32-bit blocks*/
    public byte[] blockify(String str) {
        byte[] val = new byte[blocksize];
```

```

        byte[] content = str.getBytes();

        for (int i = 0; i < content.length; i++)
            val[i] = content[i];
        for (int i = content.length; i < blocksize; i++)
            val[i] = 0;

        return val;
    }

    /*Servicing read operation */
    public Response doRead(String disk, int block) throws IOException,
    ClassNotFoundException {
        Socket s = getSocket(disk);
        new ObjectOutputStream(s.getOutputStream()).writeObject(new
    Request(Request.READ, block, null));
        return (Response) new
    ObjectInputStream(s.getInputStream()).readObject();
    }

    /*Performing swap request */
    public Response doSwap(String disk, int block, byte[] value) throws
    IOException, ClassNotFoundException {
        Socket s = getSocket(disk);
        new ObjectOutputStream(s.getOutputStream()).writeObject(new
    Request(Request.SWAP, block, value));
        return (Response) new
    ObjectInputStream(s.getInputStream()).readObject();
    }

    /* Performing Add operation */
    public Response doAdd(String disk, int block, byte[] value) throws
    IOException, ClassNotFoundException {
        Socket s = getSocket(disk);
        new ObjectOutputStream(s.getOutputStream()).writeObject(new
    Request(Request.ADD, block, value));
        return (Response) new
    ObjectInputStream(s.getInputStream()).readObject();
    }

    /*Initialization of Disks */
    public Response doInit(String disk) throws IOException,
    ClassNotFoundException{
        Socket s = getSocket(disk);

```



```

        new ObjectOutputStream(s.getOutputStream()).writeObject(new
Request(Request.INIT, 0, null));
        return (Response) new
ObjectInputStream(s.getInputStream()).readObject();
    }

    /*Responding to read/writes to nuked disks */
    public Response doNuke(String disk) throws IOException,
ClassNotFoundException{
        Socket s = getSocket(disk);
        new ObjectOutputStream(s.getOutputStream()).writeObject(new
Request(Request.NUKE, 0, null));
        return (Response) new
ObjectInputStream(s.getInputStream()).readObject();

    }

    /*Recovering nuked Disks */
    public Response doRecover(String disk) throws IOException,
ClassNotFoundException {
        Socket[] socks = new Socket[disks];
        ObjectOutputStream[] oo = new ObjectOutputStream[disks];
        ObjectInputStream[] io = new ObjectInputStream[disks];

        int culprit=0;

        Response res = null;

        for (int i = 0; i < disks; i++) {
            String diskname = "disk" + (i+1);
            System.out.println(disk + "<=>" + diskname);
            if (disk.equals(diskname)) {
                System.out.println("culprit is :"+diskname);
                culprit=i;
                break;
            }
        }

        socks[culprit] = getSocket(disk);

        System.out.println("Culprit: " + culprit);
        for (int j = 0; j < blocks; j++) {
            System.out.println("Recovering block: " + j);
            byte[] sum = new byte[blocksize];
            for (int i = 0; i < disks; i++) {
                System.out.println("Reading from disk: " + i);
                if (i==culprit)

```

```

        continue;
        String diskname = "disk" + (i+1);
        if (socks[i] == null)
            socks[i] = getSocket(diskname);

        if (oo[i] == null)
            oo[i] = new
ObjectOutputStream(socks[i].getOutputStream());

        oo[i].writeObject(new Request(Request.READ, j,
null));
        if (io[i] == null)
            io[i] = new
ObjectInputStream(socks[i].getInputStream());
        res = (Response) io[i].readObject();
        if (res.status != Response.OK) {
            System.err.println("Too many disks
crashed...giving up!");
            System.exit(1);
        }
        applyxor(sum, res.value);
    }

    if (oo[culprit] == null)
        oo[culprit] = new
ObjectOutputStream(socks[culprit].getOutputStream());

    oo[culprit].writeObject(new Request(Request.WRITE, j,
sum));

    if (io[culprit] == null)
        io[culprit] = new
ObjectInputStream(socks[culprit].getInputStream());

    res = (Response) io[culprit].readObject();
    if (res.status != Response.OK) {
        System.err.println("Too many disks
crashed...giving up!");
        System.exit(1);
    }
}

oo[culprit].writeObject(new Request(Request.RECOVER, 0,
null));
res = (Response) io[culprit].readObject();
if (res.status != Response.OK) {

```

```
        System.err.println("Too many disks crashed...giving up!");
        System.exit(1);
    }

    return res;
}

/*Printing appropriate output to user */
public String printBytes(byte[] value) {
    int columns=32, column=0;

    StringWriter out = new StringWriter();

    for (int i = 0; i < value.length; i++) {
        int b = (int) value[i];
        char c = (char) b;

        if (b >= 0x20 && b < 0x7e)
            out.write((int)c);
        else
            out.write("\\\" + Integer.toHexString(b));

        if (column++ >= 32) {
            out.write("\n");
            column=0;
        }
    }
    out.write("\n");
    return out.toString();
}
}
```

Disk. Java

/* The Disk is a daemon that will sit around and service requests. We use a simple parity code with two data disks and one parity disk. It will respond to read/write requests from clients and also to requests for swap and add. The write command will use swap and add to update the disk. When a disk is nuked, it won't respond to read/write requests from clients. We should call the manager again with the recover command. Then the disk will start servicing requests again. */

```
import java.net.*;
import java.io.*;
import java.util.*;
```

```
public class Disk implements Runnable
{

    private static Properties diskProperties;

    public static final int LIVE=1;
    public static final int NUKED=0;

    static int blocksize, blocks; //bytes
    static int status;
    static RandomAccessFile file;

    private Socket s;
    private ObjectInputStream in = null;
    private ObjectOutputStream out = null;

    /* Starting a thread for each disk */
    public Disk(Socket s) throws IOException {
        this.s = s;
        new Thread(this).start();
    }

    /*Processing and doing the read operation on the specified block */
    private byte[] read(int block) throws IOException {
        int position = block*blocksize;
        byte[] val = new byte[blocksize];
        file.seek(position);
        file.read(val);
        return val;
    }
}
```

```

/* Doing the swap operation */
private byte[] swap(int block, byte[] value) throws IOException {
    int position = block*blocksize;
    byte[] old = new byte[blocksize];
    file.seek(position);
    file.read(old);
    file.seek(position);
    file.write(value);
    return old;
}

```

```

/* function to apply XOR */
private void applyxor(byte[] old, byte[] delta) {
    for (int i = 0; i < old.length; i++) {
        old[i] = (byte)(((int)old[i])^((int)delta[i]) & 0xff);
    }
}

```

```

/* Doing the add operation */
private void add(int block, byte[] delta) throws IOException {
    int position = block*blocksize;
    byte[] old = new byte[blocksize];

    file.seek(position);
    file.read(old);

    applyxor(old, delta);

    file.seek(position);
    file.write(old);
}

```

```

/* Servicing init operation and Initialization of disks to null */
private void init() throws IOException {
    byte[] zeros = new byte[blocksize];
    for (int i = 0; i < blocks; i++) {
        int position = i*blocksize;
        file.seek(position);
        file.write(zeros);
    }
    this.status = LIVE;
}

```

```

/*Servicing write requests */
private void write(int block, byte[] value) throws IOException {
    int position = block*blocksize;
    file.seek(position);
    file.write(value);
}

/* Realizing disk failure condition by setting flag */
public void nuke() {
    this.status = NUKED;
}

/*Acknowledging Recovery of nuked disks */
public void recover() {
    this.status = LIVE;
}

/* Processing each client request */
public void run() {

    try{
        while (true) {
            Request req = null;
            Response res = null;

            if (in == null)
                in = new
ObjectInputStream(s.getInputStream());

            req = (Request) in.readObject();
            if (status == NUKED && (req.type ==
Request.READ || req.type == Request.SWAP || req.type == Request.ADD)) {
                res = new Response(Response.ERROR,
null);
            } else {
                switch(req.type) {
                    case Request.READ:
                        byte[] val = read(req.block);
                        res = new
Response(Response.OK, val);
                        break;
                    case Request.SWAP:
                        byte[] old = swap(req.block,
req.value);

```

```

        res = new
Response(Response.OK, old);
        break;
    case Request.ADD:
        add(req.block, req.value);
        res = new
Response(Response.OK, null);
        break;
    case Request.INIT:
        init();
        res = new
Response(Response.OK, null);
        break;
    case Request.NUKE:
        nuke();
        res = new
Response(Response.OK, null);
        break;
    case Request.WRITE:
        write(req.block, req.value);
        res = new
Response(Response.OK, null);
        break;
    case Request.RECOVER:
        recover();
        res = new
Response(Response.OK, null);
        break;
    }
}

    if (out == null)
        out = new
ObjectOutputStream(s.getOutputStream());
    out.writeObject(res);
} catch
    (IOException e) {
        //System.out.println("Error processing request..." +
e);
    } catch (ClassNotFoundException e) {
        //System.out.println("Error processing request..." +
e);
    }
}

    try {
        s.close();
    } catch (IOException e) {

```

```

    }
}

/* Handling wrong no. of inputs by printing Usage */
public static void printUsage() {
    System.out.println("Usage: java Disk diskname");
    System.out.print("diskname: one of ");
    int disks = Integer.parseInt(diskProperties.getProperty("disks")),
blocks = Integer.parseInt(diskProperties.getProperty("blocks"));
    for (int i=0; i < disks; i++)
        System.out.print("disk" + (i+1) + " ");
    System.out.println();
}

/* The main function */
public static void main(String[] args) throws IOException {

    diskProperties = new Properties();
    diskProperties.load(new FileInputStream("disk.properties"));

    if (args.length < 1) {
        printUsage();
        System.exit(1);
    }

    String disk = args[0];
    String filename = diskProperties.getProperty(disk+".filename");
    String address = diskProperties.getProperty(disk+".address");
    int port =
Integer.parseInt(diskProperties.getProperty(disk+".port"));

    status = LIVE;
    blocksize =
Integer.parseInt(diskProperties.getProperty("blocksize"));
    blocks = Integer.parseInt(diskProperties.getProperty("blocks"));

    System.out.println("Starting with address:"+address+"
port:"+port + " file:" + filename);

    try {
        file = new RandomAccessFile(new File(filename), "rw");
    } catch (IOException e) {
        System.err.println("Cannot access file:" + filename + e);
        System.exit(1);
    }
}

```



```
    }  
  
    ServerSocket ss = null;  
    try {  
        ss = new ServerSocket(port);  
    } catch (IOException e) {  
        System.err.println("Unable to spawn server socket,  
terminating...");  
        System.exit(1);  
    }  
  
    while (true) {  
        new Disk(ss.accept());  
    }  
}
```

ClientGUI. Java

```
/*This program creates a Graphical User Interface (GUI) for the project and
hides the implementation details and the programming environment from the
user using swing. */
```

```
import java.net.*;
import java.io.*;
import java.util.Hashtable;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
import javax.swing.event.*;
import java.util.*;
import java.net.*;
import java.io.*;
import java.text.*;

public class ClientGUI implements ListSelectionListener, ActionListener {

    private Client client;
    private Properties diskProperties;

    private JList list;
    private JPanel wholePanel;
    private JTabbedPane tabbedPane;

    private JButton initButton, nukeButton, recoverButton, readButton,
writeButton;
    private JTextField blockField, valueField;

    private JPanel mainPanel;
    private JTextArea introText;

    private String mainIntro="";

    public ClientGUI(Client client, Properties diskProperties) {
        this.client = client;
        this.diskProperties = diskProperties;

        int ndisks =
Integer.parseInt(diskProperties.getProperty("disks"));
        String[] disks = new String[ndisks];
        for (int i = 1; i <= ndisks; i++)
            disks[i-1] = "disk" + i;
```

```
list = new JList(disks);
    list.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
list.addListSelectionListener(this);
    JScrollPane listScrollPane = new JScrollPane(list);

    mainPanel = new JPanel();
    mainPanel.setLayout(new BorderLayout(mainPanel,
BoxLayout.Y_AXIS));

    introText = new JTextArea(7, 26);
    Border emptyBorder = BorderFactory.createEmptyBorder();
    introText.setBorder(emptyBorder);
    JScrollPane introPanel = new JScrollPane(introText);
    introText.setLineWrap(true);
    introText.setWrapStyleWord(true);
    introText.setEditable(false);

    mainPanel.add(introPanel);

    initButton = new JButton("Initialize");
    initButton.setEnabled(false);
    initButton.addActionListener(this);

    nukeButton = new JButton("Nuke");
    nukeButton.setEnabled(false);
    nukeButton.addActionListener(this);

    recoverButton = new JButton("Recover");
    recoverButton.setEnabled(false);
    recoverButton.addActionListener(this);

    JPanel bpanel1 = new JPanel(new FlowLayout());
    bpanel1.add(initButton);
    bpanel1.add(nukeButton);
    bpanel1.add(recoverButton);

    blockField = new JTextField();
    blockField.setColumns(3);
    blockField.addActionListener(this);

    readButton = new JButton("Read");
    readButton.setEnabled(false);
    readButton.addActionListener(this);

    valueField = new JTextField();
    valueField.setEnabled(false);
    valueField.setColumns(40);
```

```

valueField.addActionListener(this);

writeButton = new JButton("Write");
writeButton.setEnabled(false);
writeButton.addActionListener(this);

JPanel bpanel2 = new JPanel(new BorderLayout());
bpanel2.add(new JLabel("Block"));
bpanel2.add(blockField);
bpanel2.add(readButton);
bpanel2.add(valueField);
bpanel2.add(writeButton);

JPanel panel = new JPanel(new BorderLayout());
panel.add(mainPanel, "North");
panel.add(bpanel1, "Center");
panel.add(bpanel2, "South");

JPanel vobPanel = new JPanel();
vobPanel.setLayout(new BorderLayout(vobPanel,
BoxLayout.X_AXIS));
vobPanel.add(listScrollPane);
vobPanel.add(panel);

tabbedPane = new JTabbedPane();
tabbedPane.addTab("Client GUI", vobPanel);
tabbedPane.setSelectedIndex(0);

wholePanel = new JPanel();
wholePanel.setLayout(new GridLayout(1, 1));
wholePanel.add(tabbedPane);
}

public JPanel getPanel() {
    return wholePanel;
}

public void valueChanged(ListSelectionEvent e) {
    if (e.getValueIsAdjusting())
        return;

    JList theList = (JList)e.getSource();
    if (theList.isSelectionEmpty()) {
        clearInputs();
    } else {
        int index = theList.getSelectedIndex();
        introText.setText(mainIntro);
    }
}

```

```

        introText.setCaretPosition(0);
        enableInputs();
    }
}

private void clearInputs() {
    list.clearSelection();
    initButton.setEnabled(false);
    nukeButton.setEnabled(false);
    recoverButton.setEnabled(false);
    blockField.setText("");
    blockField.setEnabled(false);
    readButton.setEnabled(false);
    valueField.setText("");
    valueField.setEnabled(false);
    writeButton.setEnabled(false);
}

private void enableInputs() {
    initButton.setEnabled(true);
    nukeButton.setEnabled(true);
    recoverButton.setEnabled(true);
    blockField.setText("");
    blockField.setEnabled(true);
    readButton.setEnabled(true);
    valueField.setText("");
    valueField.setEnabled(true);
    writeButton.setEnabled(true);
}

public void actionPerformed(ActionEvent e) {
    if (e.getSource() instanceof JButton) {
        JButton butt = (JButton)e.getSource();
        String disk = (String)list.getSelectedValue();

        try {
            if (butt.equals(initButton)) {
                introText.append("Initialize disk:" + disk +
"\n");
                Response res = client.doInit(disk);
                if (res.status == Response.OK) {
                    introText.append("Disk said OK\n");
                } else {
                    introText.append("Disk is nuked!\n");
                }
            }
            } else if (butt.equals(nukeButton)) {
                introText.append("Nuke disk:" + disk + "\n");
            }
        }
    }
}

```

```

        Response res = client.doNuke(disk);
        if (res.status == Response.OK) {
            introText.append("Disk said OK\n");
        } else {
            introText.append("Disk is nuked!\n");
        }
    } else if (butt.equals(recoverButton)) {
        introText.append("Recover disk:" + disk +
"\n");

        Response res = client.doRecover(disk);
        if (res.status == Response.OK) {
            introText.append("Disk said OK\n");
        } else {
            introText.append("Disk is nuked!\n");
        }
    } else if (butt.equals(readButton)) {
        int block =
Integer.parseInt(blockField.getText());
        introText.append("Read disk:" + disk + ",
block:" + block + "\n");

        Response res = client.doRead(disk, block);
        if (res.status == Response.OK) {
            introText.append("Disk returned the
following value:\n");

            introText.append(client.printBytes(res.value));
        } else {
            introText.append("Disk is nuked!\n");
        }
    } else if (butt.equals(writeButton)) {
        int block =
Integer.parseInt(blockField.getText());
        String valustr = valueField.getText();
        introText.append("Write disk:" + disk + ",
block:" + block + ", value:" + valustr + "\n");
        byte[] value = client.blockify(valustr);

        Response res = client.doSwap(disk, block,
value);

        if (res.status == Response.OK) {
            introText.append("Disk said OK\n");
            client.applyxor(value, res.value);
            String parity =
diskProperties.getProperty("paritydisk");
            res = client.doAdd(parity, block,
value);

            if (res.status == Response.OK) {

```

```

        introText.append("Parity disk
said OK\n");
        } else {
            introText.append("Parity disk
is nuked!\n");
        }
    } else {
        introText.append("Disk is nuked!\n");
    }
}
} catch (IOException ioex) {
    System.err.println(ioex);
} catch (ClassNotFoundException cnfex) {
    System.err.println(cnfex);
}
clearInputs();
}
}

```

```

public static void main(String[] args) throws IOException,
ClassNotFoundException {

    Properties diskProperties = new Properties();
    diskProperties.load(new FileInputStream("disk.properties"));
    Client client = new Client(diskProperties);

    JFrame frame = new JFrame("STARFISH GUI");
    frame.addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {System.exit(0);}
    });

    ClientGUI clientGUI = new ClientGUI(client, diskProperties);
    frame.getContentPane().add(clientGUI.getPanel());
    frame.pack();
    frame.setResizable(false);
    frame.setVisible(true);
}
}

```

10.2 Sample output (at the command prompt)

Invoking the Disk Daemons-Disk1, Disk2, Disk3:

```
C:\Project>javac Disk.java
```

```
C:\Project>java Disk
Usage: java Disk diskname
diskname: one of disk1 disk2 disk3
```

```
C:\Project>java Disk disk1
Starting with address:localhost port:1234 file:disk1
```

```
C:\Project>java Disk disk2
Starting with address:localhost port:2345 file:disk2
```

```
C:\Project>java Disk disk3
Starting with address:localhost port:3456 file:disk3
```

Running and using the Client Program:

```
C:\Project>javac Client.java
```

```
C:\Project>java Client
Usage: java Client diskname operation block [value]
diskname: one of disk1 disk2 disk3
operation: one of read write init nuke recover
block (in the case of read/write): from 0 to 99
value (in the case of write): any string (use double quotes if spaces present)
```

```
C:\Project>java Client disk1 init
Disk said OK
```

```
C:\Project>java Client disk2 init
Disk said OK
```

```
C:\Project>java Client disk3 init
Disk said OK
```

```
C:\Project>java Client disk1 write 11 100
Disk said OK
Parity disk said OK
```

```
C:\Project>java Client disk2 write 11 111
Disk said OK
Parity disk said OK
```


C:\Project>java Client disk1 read 11

Disk said OK

1 0
0 0 0 0 0 0

C:\Project>java Client disk2 read 11

Disk said OK

1 1 1 0
0 0 0 0 0 0

C:\Project>java Client disk3 read 11

Disk said OK

0 1 1 0
0 0 0 0 0 0

C:\Project>java Client disk1 nuke

Disk said OK

C:\Project>java Client disk1 read 11

Disk is nuked!

C:\Project>java Client disk1 recover

disk1<=>disk1

culprit is :disk1

Recovering block: 0

Reading from disk: 0

Reading from disk: 1

Reading from disk: 2

Recovering block: 1

Reading from disk: 0

Reading from disk: 1

Reading from disk: 2

Recovering block: 2

Reading from disk: 0

Reading from disk: 1

Reading from disk: 2

Recovering block: 3

Reading from disk: 0

Reading from disk: 1

Recovering block: 3

Reading from disk: 0

Reading from disk: 1

Reading from disk: 2

Reading from disk: 2

Recovering block: 3

Reading from disk: 0

Reading from disk: 1

Reading from disk: 2

...

...

...

...

Recovering block: 99

Reading from disk: 0

Reading from disk: 1

Reading from disk: 2

Disk said OK

C:\Project>java Client disk1 read 11

Disk said OK

1 0
0 0 0 0 0 0

C:\Project>