**Distributed Storage System (DSS) Application Design Documentation**

**1. Message Format Design**

**1.1 General Message Format**

All messages in our DSS implementation follow a consistent format:

[LENGTH\_PREFIX (4 bytes)][MESSAGE\_PAYLOAD]

* **Length Prefix**: 4-byte unsigned integer in network byte order indicating payload length
* **Message Payload**: UTF-8 encoded string with space-separated components

**1.2 Command-Specific Message Formats**

**Manager Protocol Messages**

**1. register-user**

Format: "register-user <user-name> <IPv4-addr> <m-port> <c-port>"

Example: "register-user Alice 127.0.0.1 93511 93512"

Response: "SUCCESS" or "FAILURE"

**2. register-disk**

Format: "register-disk <disk-name> <IPv4-addr> <m-port> <c-port>"

Example: "register-disk D1 127.0.0.1 93501 93502"

Response: "SUCCESS" or "FAILURE"

**3. configure-dss**

Format: "configure-dss <dss-name> <n> <striping-unit>"

Example: "configure-dss DSS1 5 1024"

Response: "SUCCESS" or "FAILURE"

**4. ls (list files)**

Format: "ls"

Response: "SUCCESS\nDSS1: Disk array with n=5 (D1, D2, D3, D4, D5) with striping-unit 1024 B.\n file1.txt 2048 B User1"

**5. copy**

Format: "copy <file-name> <file-size> <owner>"

Example: "copy document.txt 1024 Alice"

Response: "SUCCESS <dss-name> <file-size> <n> <striping-unit> <num-disks> <disk0-info> <disk1-info> ..."

**6. read**

Format: "read <dss-name> <file-name> <user-name>"

Example: "read DSS1 document.txt Alice"

Response: "SUCCESS <file-size> <n> <striping-unit> <num-disks> <disk0-info> <disk1-info> ..."

**7. disk-failure**

Format: "disk-failure <dss-name>"

Example: "disk-failure DSS1"

Response: "SUCCESS" or "FAILURE"

**8. copy-complete**

Format: "copy-complete <dss-name> <file-name> <owner>"

Example: "copy-complete DSS1 document.txt Alice"

Response: "SUCCESS"

**9. read-complete**

Format: "read-complete <dss-name> <file-name> <user-name>"

Example: "read-complete DSS1 document.txt Alice"

Response: "SUCCESS"

**10. recovery-complete**

Format: "recovery-complete <dss-name>"

Example: "recovery-complete DSS1"

Response: "SUCCESS"

**Peer-to-Peer (P2P) Protocol Messages**

**1. store-block**

Format: "store-block <dss-name> <file-name> <stripe-num> <block-type> <data-size>"

Example: "store-block DSS1 file1.txt 0 data 1024"

Response: "SUCCESS" or "FAILURE"

**2. read-block**

Format: "read-block <dss-name> <file-name> <stripe-num>"

Example: "read-block DSS1 file1.txt 0"

Response: "SUCCESS <data-size> <block-type>" or "FAILURE"

**3. fail**

Format: "fail"

Response: "FAIL-COMPLETE"

**4. get-stripe**

Format: "get-stripe <dss-name> <file-name> <stripe-num>"

Example: "get-stripe DSS1 file1.txt 0"

Response: "SUCCESS <block-type> <data-size>" or "FAILURE"

**2. Time-Space Diagrams**

**2.1 User Registration Sequence**

User Manager

| |

|--- register-user ------>|

| |--- Validate parameters

| |--- Store user info

|<------ SUCCESS ---------|

| |

**2.2 Disk Registration Sequence**

Disk Manager

| |

|--- register-disk ------>|

| |--- Validate parameters

| |--- Store disk info

| |--- Set state to "Free"

|<------ SUCCESS ---------|

| |

**2.3 DSS Configuration Sequence**

User Manager

| |

|--- configure-dss ------>|

| |--- Validate n >= 3

| |--- Check striping unit

| |--- Select n free disks

| |--- Update disk states

| |--- Store DSS config

|<------ SUCCESS ---------|

| |

**2.4 File Copy Operation Sequence**

User Manager Disk1 Disk2 Disk3

| | | | |

|--- copy req ------->| | | |

| |--- select DSS ---| | |

|<-- DSS params ------| | | |

| | | | |

|--- store-block -----|------------------>| | |

| | |--- store data | |

|<--- SUCCESS --------|-------------------| | |

| | | | |

|--- store-block -----|-------------------------->| | |

| | | |--- store data |

|<--- SUCCESS --------|---------------------------| | |

| | | | | |

|--- store-block -----|-------------------------------------->| |

| | | | |--- store parity

|<--- SUCCESS --------|---------------------------------------| |

| | | | | |

|--- copy-complete -->| | | | |

|<--- SUCCESS --------| | | | |

**2.5 File Read Operation Sequence**

User Manager Disk1 Disk2 Disk3

| | | | |

|--- read req ------->| | | |

| |--- verify file --| | |

|<-- file params -----| | | |

| | | | |

|--- read-block ------|------------------>| | |

| | |--- read data | |

|<--- block data -----|-------------------| | |

| | | | |

|--- read-block ------|-------------------------->| | |

| | | |--- read data |

|<--- block data -----|---------------------------| | |

| | | | | |

|--- read-complete -->| | | | |

|<--- SUCCESS --------| | | | |

**2.6 Disk Failure and Recovery Sequence**

User Manager Disk1 Disk2 Disk3

| | | | |

|--- disk-failure --->| | | |

| |--- select disk --| | |

| |--- send fail -----|-------------->| |

| | |--- set failed | |

| |<-- fail-complete--|---------------| |

|<--- SUCCESS --------| | | |

| | | | |

|--- recovery req --->| | | |

| |--- get-stripe ----|-------------------------->| |

| | | |<-- data ------|

| |--- get-stripe ----|-----> | | |

| | |<-- data ------| |

| |--- XOR recover ---| | |

| |--- store recovered|-------------->| |

| |<-- success -------|---------------| |

|<-- recovery-comp ---| | | |

**3. Data Structures and Design Decisions**

**3.1 Manager State Information**

The manager maintains several key data structures:

class DSS\_Manager:

def \_\_init\_\_(self):

# User registry: user\_name -> user\_info

self.users = {

'user\_name': {

'address': 'IPv4\_address',

'm\_port': int,

'c\_port': int

}

}

# Disk registry: disk\_name -> disk\_info

self.disks = {

'disk\_name': {

'address': 'IPv4\_address',

'm\_port': int,

'c\_port': int,

'state': 'Free|InDSS|Failed'

}

}

# DSS configurations: dss\_name -> config

self.dss\_configs = {

'dss\_name': {

'n': int, # Number of disks

'striping\_unit': int, # Block size in bytes

'disk\_order': [disk\_names] # Ordered list of disk names

}

}

# File metadata: dss\_name -> files

self.files = {

'dss\_name': {

'file\_name': {

'size': int, # File size in bytes

'owner': 'user\_name' # File owner

}

}

}

# Operation tracking

self.read\_operations = {} # Track ongoing reads

self.lock = threading.Lock() # Thread synchronization

**3.2 Disk State Information**

Each disk process maintains:

class DSS\_Disk:

def \_\_init\_\_(self):

# Block storage: file\_identifier -> stripe\_data

self.storage = {

'dss\_name\_file\_name': {

'stripes': {

'stripe\_number': {

'type': 'data|parity',

'data': 'hex\_encoded\_bytes',

'size': int

}

}

}

}

# DSS membership info

self.dss\_info = {} # DSS configuration cache

self.failed = False # Failure simulation state

self.lock = threading.Lock() # Thread synchronization

**3.3 User State Information**

User processes maintain minimal state:

class DSS\_User:

def \_\_init\_\_(self):

# Connection information only

self.user\_name = str

self.manager\_ip = str

self.manager\_port = int

self.m\_port = int # Management port

self.c\_port = int # Command port

self.lock = threading.Lock() # Thread synchronization

**3.4 Key Design Decisions**

**3.4.1 Block-Interleaved Distributed Parity (BIDP) Implementation**

* **Stripe Distribution**: Parity blocks are distributed across disks using the formula: parity\_disk = (n - 1 - (stripe\_num % n)) % n
* **Block Size**: All blocks padded to striping\_unit size with null bytes
* **XOR Parity**: Simple XOR operation across all data blocks in a stripe

**3.4.2 Message Protocol Design**

* **Length-Prefixed Messages**: Ensures reliable message boundaries over TCP
* **Space-Separated Format**: Simple parsing while maintaining readability
* **Synchronous Communication**: Each command waits for response before proceeding

**3.4.3 Concurrency Design**

* **Multi-threaded Servers**: Each process handles multiple concurrent connections
* **Thread-Safe State**: All shared data structures protected by locks
* **Separate Ports**: Management and command operations use different ports

**3.4.4 Error Handling Strategy**

* **Graceful Degradation**: Operations fail cleanly with appropriate error codes
* **State Consistency**: Failed operations don't leave system in inconsistent state
* **Recovery Support**: Failed disks can be recovered using parity reconstruction

**3.4.5 Storage Implementation**

* **File-Based Storage**: Each disk stores data in JSON files for simplicity
* **Hex Encoding**: Binary data stored as hex strings for JSON compatibility
* **Hierarchical Organization**: Files organized by DSS and file name

**3.5 State Transitions**

**Disk States**

Free → InDSS → [Failed] → [Recovered] → Free

**File Operation States**

Copy: Request → Stripe Processing → Storage → Complete

Read: Request → Stripe Assembly → Verification → Complete

Recovery: Failure → Analysis → Reconstruction → Restoration

**4. Implementation Details**

**4.1 Threading Architecture**

* **Manager**: Main thread accepts connections, worker threads handle clients
* **Disk**: Command server thread accepts P2P connections, worker threads process requests
* **User**: Single-threaded with synchronous operations for simplicity

**4.2 Network Communication**

* **TCP Sockets**: Reliable, ordered delivery for all communications
* **Connection Management**: New connection per operation (stateless design)
* **Error Recovery**: Socket errors handled with appropriate cleanup

**4.3 BIDP Stripe Calculation**

For a file of size F bytes with n disks and striping unit S:

* Data blocks per stripe: n - 1
* Data per stripe: (n - 1) × S
* Number of stripes: ⌈F / ((n - 1) × S)⌉
* Parity disk for stripe i: (n - 1 - (i % n)) % n

**4.4 Fault Tolerance**

* **Single Disk Failure**: Can be recovered using XOR of remaining blocks
* **Read Operations**: Continue if all data blocks available
* **Write Operations**: Must have all disks available

**5. Video Demo Information**

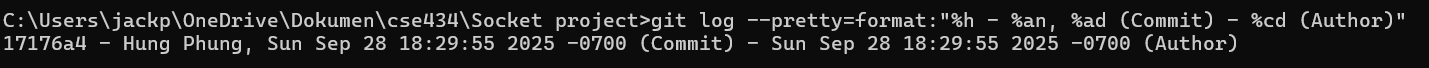
**5.1 Video Demo Link**

**[https://youtu.be/en1wY9a91T8]**

**5.3 Required GitHub Screenshots**

Please include the following screenshots from your GitHub repository:

1. **Git Log Output**: Screenshot of git log --pretty=format:"%h - %an, %ad (Commit) - %cd (Author)"

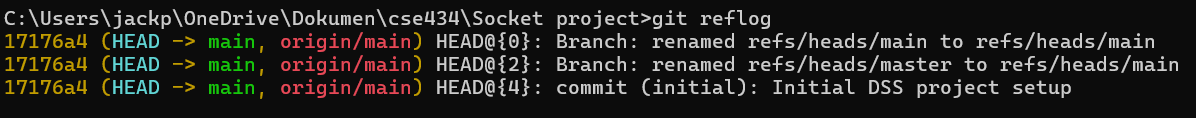


1. **Commit History**: Screenshot of entire commit history in GitHub web interface

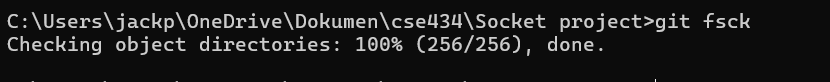
A screenshot of a computer

AI-generated content may be incorrect.

1. **Git Reflog**: Screenshot of git reflog output



1. **Git Fsck**: Screenshot of git fsck output



**6. Testing and Validation**

**6.1 Test Cases Implemented**

1. **Basic Registration**: Users and disks register successfully
2. **DSS Configuration**: Automatic DSS setup with proper disk selection
3. **File Operations**: Copy and read operations work correctly
4. **Concurrent Access**: Multiple users can operate simultaneously
5. **Error Handling**: Invalid commands return appropriate failures
6. **Failure Simulation**: Disk failures are properly simulated

**6.2 Performance Considerations**

* **Scalability**: System supports multiple DSSs and concurrent operations
* **Efficiency**: Minimal network overhead with compact message formats
* **Reliability**: All operations are atomic and error-recoverable