Implementation of Distributed File Sharing System Using Peer-to-Peer Networks

COEN 317 - Distributed Systems

Haomin Cheng
hcheng5@scu.edu
SCU ID: W1651306
Computer Science
Santa Clara University

Manasa Madiraju
mmadiraju@scu.edu
SCU ID: 07700009942
Computer Science
Santa Clara University

Sana HayatKhan Pathan
spathan@scu.edu
SCU ID: W1650177
Computer Science
Santa Clara University

Abstract - This paper delves into the intricacies of designing and implementing a robust distributed file sharing system within a peer-to-peer (P2P) network framework. By incorporating advanced algorithms, including resource discovery, leader election, mutual exclusion, concurrency, replication and timestamps, this study aims to enhance system efficiency, fault tolerance, and scalability. A thorough exploration of the project's methodology, implementation, and algorithmic performance evaluation contributes valuable insights to the field of distributed computing.

Keywords - Peer-to-Peer, Distributed File Sharing, Resource Discovery, Leader Election, Mutual Exclusion, Concurrency, Replication, Timestamps.

1. INTRODUCTION

In the ever-evolving landscape of information sharing, the limitations of centralized file-sharing models have prompted an exploration of decentralized architectures. This project endeavors to address these challenges by implementing a sophisticated peer-to-peer distributed file sharing system. We are implementing it for the class syllabus file sharing among peers, so it's a syllabus sharing application. The objective of our project is to create a decentralized P2P file sharing system where a central node stores connection information and facilitates distributed file searches among connected peers; This approach is beneficial to not only overcome the limitations of centralized file-sharing models but also improves scalability and fault tolerance amongst many other benefits.

The functionality here is to allow users to share files directly with one another, notifying peers if the requested file exists in the network and support the dynamic addition and removal of peers along with dynamically maintaining the files.

In our project, we have tried to address the challenges faced in distributed file systems, including data consistency, concurrency, heterogeneity, security, and performance overhead.

Using a distributed file system can pose several challenges, including ensuring data consistency across nodes, managing fault tolerance, addressing security concerns, handling scalability issues, minimizing performance overhead, and dealing with data fragmentation. Overcoming these challenges requires implementing strategies such as data replication, fault-tolerant architectures, robust security protocols, efficient load balancing, and streamlined data retrieval mechanisms. By addressing these issues effectively, distributed file systems can operate seamlessly and reliably, accommodating a growing volume of data and users while maintaining optimal performance and data integrity.

2. MOTIVATION

Our project is driven by the need for a decentralized, efficient, and scalable file-sharing solution. Traditional file-sharing platforms often face limitations such as restricted storage capacity and centralized control, which can hinder user experiences and data management.

Our goal is to address these challenges by creating a platform that empowers users to have control over their data through a decentralized storage architecture. This approach aims to streamline access to user data, ensuring a seamless and efficient sharing process, and providing scalability for easy expansion and adaptability to meet users' dynamic demands.

We aspire to implement the P2P sharing model, enhancing flexibility and scalability, and allowing peers to join or leave the network seamlessly without disrupting overall operations. The central node in our design plays a crucial role in efficient connection management and file searches among connected peers.

Ultimately, our motivation is to develop a robust, user-friendly file-sharing platform that allows users to share and access files with ease, maintaining ownership and control over their data in a dynamic P2P environment.

3. LITERATURE REVIEW

File Sharing Strategy Based on WebRTC[9]: The paper proposes a P2P file-sharing strategy that enhances the efficiency of file transfers by slicing files into blocks and forwarding them through browsers using WebRTC. Each block is sent to a specific node one by one. When a peer finishes receiving a block, it forwards the data to other receivers. This approach aims to utilize free nodes that have completed receiving, thereby increasing overall data throughput and improving transmission rates. The proposed strategy can improve the efficiency of file transfers in a P2P network by utilizing the capabilities of free nodes, reducing waiting times, and improving transmission rates. Implementing file slicing into blocks for more manageable and efficient file sharing, allowing for parallel transmission and reception of file parts. In the context of future work, the authors advocate for the integration of a network detection module. This enhancement is projected to obviate the impediments posed by the suboptimal performance of individual workers, thereby optimizing the overall file transfer speed and enhancing the robustness and efficiency of the P2P file-sharing system. The methodologies and considerations expounded in this work pave the way for the development of optimized P2P file-sharing systems, marked by enhanced transmission rates and the secure, direct communication facilitated by WebRTC.

P2P Application for File Sharing[5]: The paper presents a centralized P2P application for file sharing, evaluating its performance through simulated experiments. It starts by discussing the key principles of the P2P model, highlighting the collaborative nature among users and the resulting benefits in network reliability, service capacity, and flexibility. The paper categorizes P2P file-sharing applications into centralized/hybrid or decentralized/pure architectures and structured and unstructured networks based on how indexing information is managed. The proposed system architecture includes a directory server managing a list of peers and their content, shared content sources, and peers responsible for relaying and receiving content. An innovative algorithm is introduced for creating and maintaining the overlay network to minimize delays and optimize resource utilization. The performance evaluation focuses on the relationship between file size, available link bandwidth, and the number of connected peers, demonstrating their impact on file retrieval time. The paper concludes by underlining the advantages of the P2P application over the client-server model and suggesting future research directions, such as incorporating video streaming applications and exploring methods to enhance system scalability and eliminate single points of failure by removing the directory server.

Efficient File Sharing Strategy in DHT Based P2P Systems[6]: The paper focuses on the challenges and solutions within P2P file-sharing systems, highlighting the importance of efficient routing and retrieval algorithms. It proposes a hierarchical routing and retrieval algorithm designed to improve the shortcomings of DHT algorithms by utilizing peers' topological information to locate the closest copy for any routing request, thereby enhancing system performance. Emphasizing the decentralized and cooperative nature of P2P systems, it contrasts them with traditional Client-Server architectures and underscores the significance of features like decentralized control, self-organization, fault tolerance, and load balancing. The paper thoroughly discusses the operations in P2P systems, particularly the routing and file retrieval processes, and examines various DHT-based routing algorithms, including Chord, Pastry, Tapestry, and CAN, outlining their roles in establishing an efficient infrastructure for P2P systems.

Additionally, it addresses the challenges faced by these algorithms, focusing on the discrepancy between logical routing structures and the physical distribution of peers in real-world environments. Overall, the paper aims to introduce and validate a hierarchical routing and retrieval algorithm that can significantly enhance the performance of P2P file-sharing systems by considering the complexities of network connections, diverse peers, and varying service requirements.

Cost Effective File Replication in P2P File Sharing Systems[3]: The authors delve into the realm of P2P file-sharing systems, particularly aiming to enhance the file replication process. Their primary goal is to ensure data availability in P2P networks while concurrently minimizing associated expenses. To achieve this objective, the authors explore diverse replication strategies, emphasizing the improvement of data redundancy, reduction in network traffic, and strengthening of fault tolerance mechanisms. Notably, the paper introduces economic considerations into the replication process, taking into account factors such as the cost of replication, network bandwidth utilization, and storage resource allocation. This economic perspective gives rise to the proposal of an economically efficient replication strategy that optimizes data replication while minimizing the impact on network resources. Through the use of simulations, the authors assess the effectiveness of these proposed strategies, offering empirical evidence of their ability to reduce replication costs while maintaining data availability in P2P file-sharing systems.

Building Peer-to-Peer Systems With Chord, a Distributed Lookup Service[2]: The paper introduces the Chord protocol, a DHT system that revolutionizes decentralized data retrieval in large networks. Chord utilizes consistent hashing to distribute data among network nodes. Each node is assigned a specific range in the hash space, allowing data to be located by efficiently routing through nodes based on their hash values. This structured overlay network ensures effective routing and lookup, accommodating nodes' dynamic addition and departure without compromising performance. Chord achieves fault tolerance by replicating data, ensuring data availability even in node failures. Load balancing is another vital feature, preventing resource hotspots and promoting equitable data access. Beyond these technical aspects, Chord provides a versatile foundation for various P2P systems, significantly contributing to decentralized data storage and retrieval.

Distributed File Sharing Management[4]: The paper addresses challenges in efficient and scalable file sharing in a distributed environment. It utilizes distributed computing principles to create a decentralized architecture, enhancing scalability and fault tolerance. The paper introduces file replication for data redundancy and emphasizes load balancing to evenly distribute resources. Additionally, it explores efficient file search and retrieval methods, promising users swift access to files without straining the network. This research offers an effective solution to file sharing challenges within a distributed environment.

4. DESCRIPTION OF PROJECT

This project is focused on the development of a distributed file-sharing system using peer-to-peer (P2P) networks designed to manage and facilitate the sharing of course-related files. The system is implemented using Flask and handles file uploads, queries, and other operations. The primary goal is to create a decentralized and fault-tolerant system that allows users to share files across a network of interconnected nodes. The project employs various algorithms and mechanisms to address key challenges in distributed systems, ensuring reliability, scalability, and data consistency.

5. ALGORITHMS USED

The distributed file-sharing system is architected as an interconnected network of peers, each contributing to the collaborative sharing of files in a decentralized manner. This section provides an in-depth examination of the overall system structure and the fundamental principles guiding its operation.

- Resource Discovery: This algorithm facilitates the seamless integration of new nodes into the existing peer-to-peer network. It relies on a designated bootstrap node or server that maintains information about the IP and port details of all connected nodes. When a new node wants to join the network, it contacts the bootstrap node to obtain a list of currently connected nodes, enabling a smooth assimilation process.

- Leader Election: The leader election algorithm dynamically selects a leader node within the distributed system. The leader is crucial for coordinating activities and maintaining system stability. Nodes engage in periodic heartbeat exchanges to monitor the current leader's health. If the leader fails, a distributed leader election protocol is initiated, allowing nodes to collaboratively choose a new leader and ensure continuous coordination within the system.

- Mutual Exclusion: Mutual exclusion is essential for preventing conflicts when multiple nodes attempt to write to a shared resource simultaneously. This algorithm employs a distributed lock mechanism. Before performing write operations on shared resources, nodes must acquire the lock. This ensures that only one node can modify the resource at any given time, preventing inconsistencies and maintaining the integrity of shared data.

- Concurrency: This algorithm is crucial for managing concurrent access to critical sections of code, such as those responsible for updating important data structures. It utilizes locks to control access, allowing only one thread or process at a time to make modifications. This ensures that changes to essential data structures occur sequentially, avoiding conflicts and maintaining system stability.

- Replication: Replication enhances fault tolerance and availability by distributing copies of files across the network. When a node hosts a file, it replicates the file to its neighbors in a clockwise manner. This proactive replication strategy ensures that multiple copies of the file exist in the network, contributing to resilience against node failures or disruptions.

- Timestamps: Timestamps are used to maintain the order of events and ensure consistency across distributed nodes. The algorithm strategically employs timestamps to mark the occurrence of events, establishing a chronological framework. A reliable clock synchronization mechanism is implemented to ensure accurate timestamps across nodes, contributing to a unified understanding of temporal relationships within the system.

6. IMPLEMENTATION OF ALGORITHMS

This section delves into the practical aspects of how we implement the specified algorithms, providing an overview of the coding process, integration into the file-sharing system, and considerations for algorithmic efficiency and robustness. It details the steps taken to ensure seamless cohesion between the algorithms and the broader system architecture.

Resource Discovery: In the resource discovery phase of this project, a bootstrap node or server serves as a central point for storing information about connected nodes in the network. When a new node joins the system, it communicates with the bootstrap node to obtain a list of currently connected nodes. This list is crucial for the new node to integrate itself into the existing peer-to-peer network. The bootstrap node's responsibility is to maintain an updated and accurate record of the network's topology, facilitating the smooth addition of new nodes.

Leader Election: Originally, the bootstrap node assumes the role of the leader, and the supernode holds a crucial position in the application. The presence of a supernode is vital, and in the event of a supernode failure, a new supernode is elected. When the supernode experiences a shutdown, it initiates the leader election algorithm. Additionally, a separate thread runs on all server instances, regularly checking the activity status of the current leader. If the supernode exits unexpectedly, the other server instances detect the absence of the current leader and initiate a new leader election based on the alive file created after each server instance starts. The selection process involves determining the minimum value based on the calculated ID for each instance.

Mutual Exclusion:.Maintaining mutual exclusion proves indispensable in averting conflicts arising from concurrent attempts by multiple nodes to write to a shared resource. In our system implementation, we adopt a sophisticated distributed lock mechanism as a strategic measure. Prior to engaging in write operations, nodes formally request the lock, enforcing a singular node's exclusive modification rights to the shared resource. Following the completion of its operation, the node responsibly relinquishes the lock, facilitating subsequent access by other nodes. This meticulous protocol not only ensures the integrity of the shared resource but also enables multiple nodes to access it in a sequential and organized manner.

Concurrency: To manage concurrency effectively, locks are utilized to control access to critical sections of code, such as those responsible for updating neighbor or finger tables. These locks ensure that only one thread or process at a time can make changes to these tables, avoiding potential inconsistencies that could arise from concurrent modifications. The careful orchestration of concurrency ensures the proper functioning of the distributed system and prevents conflicts in the handling of essential data structures.

Replication: In the replication aspect of your project, files hosted by a node are replicated to its k neighbors in the Chord ring in a clockwise manner. This replication strategy enhances fault tolerance and availability by ensuring that multiple copies of a file exist in the network. Mechanisms are implemented to handle replication consistency, which may involve strategies such as eventual consistency or strong consistency, depending on the specific requirements of your distributed file-sharing system.

Timestamps: Timestamps play a crucial role in maintaining the order of events and ensuring consistency across distributed nodes. In the implementation, timestamps are used to mark the occurrence of events, providing a basis for establishing a global order in the system. To achieve accurate timestamps, a reliable clock synchronization mechanism is implemented, allowing nodes to maintain a consistent understanding of time. This synchronization ensures that events are appropriately ordered, contributing to the overall coherence of the distributed file-sharing system.

7. ARCHITECTURE OF THE SYSTEM

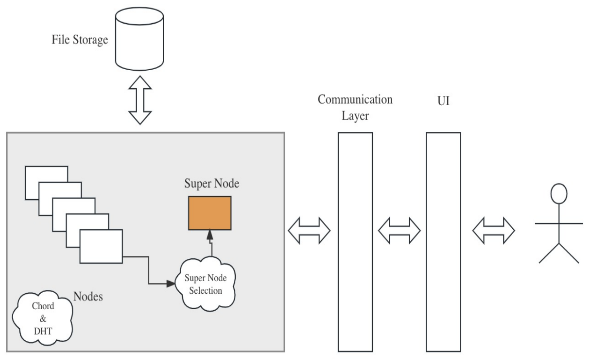


Figure : 7.1 High Level Design

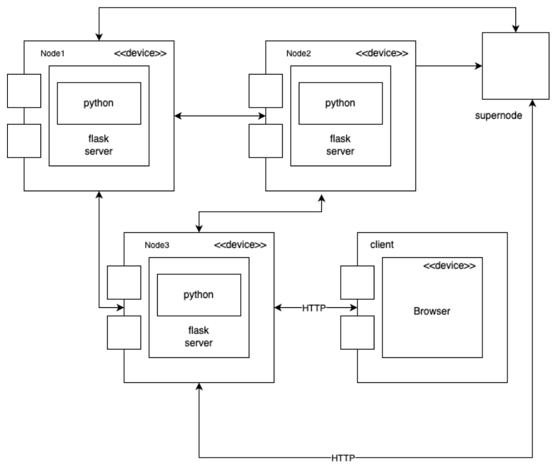


Figure 7.2 : Architecture Diagram

To explain the architecture, initially we start the server node, which is elected as a leader as its first server to connect. The leader also updates its information on leader_config.json, which resides on S3 location. This location is accessible to all the running nodes including server instances and node instances. There can be several server instances, in case the current leader goes

down.The scenario helps us with having an option for failover. Here the leader election is done on the basis of the ip and the port the server is running. So based on the ip and port an id is calculated and each server creates the copy of alive file on S3 location which is accessible to all the nodes in the network. So how does the leader election works. There is a background thread running which constantly checks from each server instance if the current leader is active or not. In case if the current leader is found inactive based on the calculated id and the alive files present on the S3 location the new leader is elected. To avoid re-electing the same supernode which went down, whenever the supernode goes down or for that matter whenever the server goes down, it deletes its own alive file, so there are no false files present in the S3 location. Another major implementation is whenever the server is gracefully terminated, it starts the leader election when going down after deleting its alive file.
Next, we start several nodes that are connected in the network who will host the files in the network for transfer. Once the node joins the supernode, it creates the finger table when the first node joins the network. Later when another node joins the finger table is updated at every event of nodes joining. Once the finger table is updated it returns with the response of the current predecessor and successor of the joined node.
In case the node leaves the network, the finger table gets updated and so does the predecessor and successor of the neighboring node.
The client application i.e the web application is running on all the node instances and can be opened using the localhost from any of the instances. Here the

8. IMPLEMENTATION DETAILS

Implementation over distributed system :

RESOURCES USED:

1. AWS EC2 instances for server and clients
2. Python for Programming Language
3. AWS S3 for object storage
4. Github
5. IEEE papers
6. React
7. Flask
8. Mantine

1) Architectural Style:

The chosen architectural style for this distributed file-sharing system is a combination of Peer-to-Peer (P2P) and Client-Server architecture.

- Peer-to-Peer (P2P): The P2P architecture is suitable for a decentralized file-sharing system. Nodes in the network act both as clients and servers, sharing files directly with each other without relying on a central server for all transactions.
- Client-Server: The leader election and resource discovery mechanisms may have a more centralized aspect where a designated bootstrap node initially plays the role of a server, providing information to new nodes. The leader node also acts as a server for coordination purposes.

2) Class Diagram

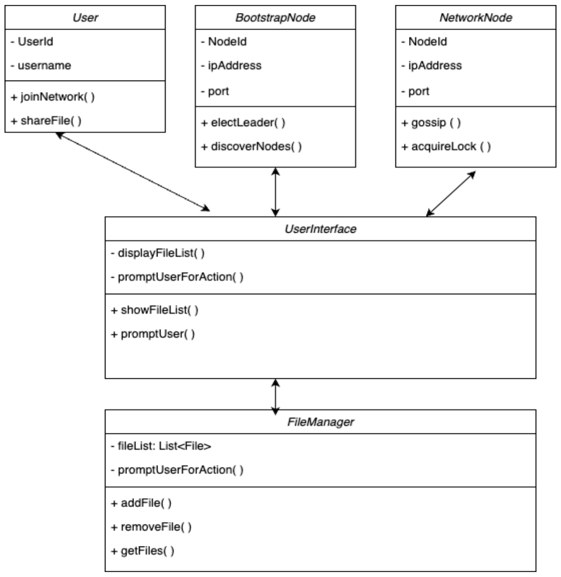


Figure 8.1 : Class Diagram

Sequence Diagram:

The sequence diagram in figure 8.2 is one of the functions that is provided to the user. So users have a web to explore the functionalities provided to them. The sequence diagram in figure 8.2 explains the sequence of the activities performed.

So to explain it in brief, the user uploads the file from a web page it wants to host on the network. The web is developed using react and can be run on any of the client machines. The request then goes to localhost.
Localhost then sends the request to supernode for the computation of the hash value for the file. Based on the hash value of the file, the supernode gets the details of the node from the finger table which should host the uploaded file in the chord network. The supernode then replies to the request client, with the details of host server and then the client then transfers the file to host. Here along with computing the host for the file, it also computes the replication server and copies the data from host file to replication server, so that in case the host server goes down, the file is still hosted on the network.

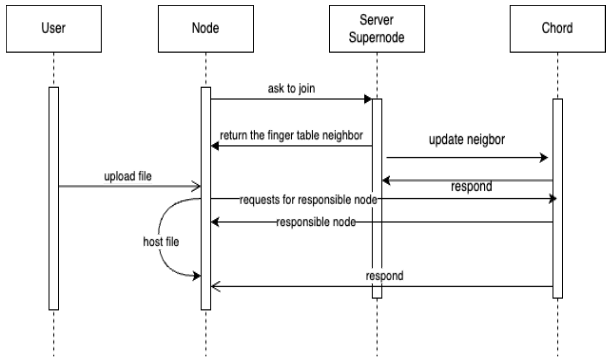


Figure 8.2 : Sequence diagram for upload

The figure 8.3 is the sequence diagram for the other functionality provided. In this, the user requests the file on the web. The request then goes to the localhost. Localhost then sends the request to supernode for the computation of the hash value for the file. Based on the hash value of the file, the supernode gets the details of the node from the finger table which should ideally be hosting the file. Once it has the details of the node which is hosting the file currently, the response is then passed from supernode to the node and then to the user. Users can then access the path to get the file.
Here we have implemented several failure scenarios, like if the host of the requested file goes down, in that case we can get the details of the replication server and it will then help the client get the file.

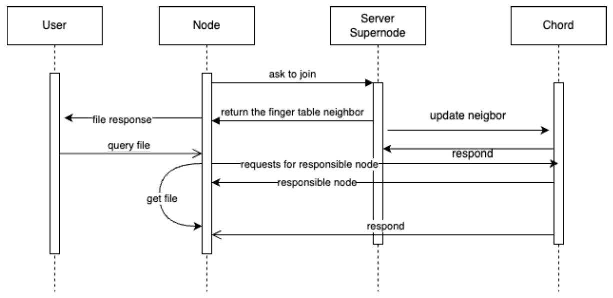


Figure 8.3 : Sequence diagram for query

3) Software Components and Services:

Communication Mechanism: RESTful APIs : Services communicate through RESTful APIs for simplicity and platform independence.

Data Storage: Distributed Database : File metadata, network node information, and leader election status are stored in a distributed database for data consistency and fault tolerance.

User Interface: Web-based Interface : User interaction occurs through a web-based interface, making it accessible across various devices.

9. KEY DESIGN GOALS ACHIEVED

In the design of distributed file-sharing systems, key goals have been meticulously crafted to address core challenges inherent in distributed systems. Each design objective is tailored to foster a robust, adaptable, and user-friendly system, capable of navigating the complexities posed by heterogeneity, openness, security, failure handling, concurrency, quality of service, scalability, and transparency.

Heterogeneity:
To enable the accommodation of a diverse range of devices and platforms within the peer-to-peer network, the resource discovery algorithm utilizes a bootstrap node/server to store connected IP and port information, providing a centralized point for heterogeneous nodes to join the network. Standardized communication protocols ensure compatibility across various devices, fostering a cohesive network environment that transcends heterogeneity.

Openness:
To facilitate seamless integration of new nodes and the dynamic evolution of the peer-to-peer network leveraging the resource discovery algorithm, the system allows for the open participation of new nodes, providing necessary information for their smooth integration. The leader election algorithm dynamically adapts to changes in leadership and network composition, ensuring an open and evolving distributed system.

Security:
To safeguard the integrity and confidentiality of data within the distributed file-sharing system, the mutual exclusion algorithm employs a distributed lock mechanism to prevent conflicts during write operations, ensuring data integrity. Security measures are integrated into the leader election and replication algorithms, fortifying system coordination and file distribution against potential security threats.

Failure Handling:
To gracefully tolerate and recover from node failures to ensure system resilience, the leader election algorithm dynamically selects new leaders in the event of leader failures, ensuring continuous coordination within the system. The replication algorithm contributes to failure handling by distributing file copies strategically, reducing the impact of node failures on data availability.

Concurrency:
To manage concurrent access to shared resources to prevent conflicts, the mutual exclusion and concurrency control algorithms use locks to coordinate access, ensuring that write operations occur sequentially. This design goal promotes data consistency and system stability in the presence of concurrent operations.

Quality of Service (QoS):
To provide reliable and predictable performance in terms of data access and system coordination. *C*oncurrency control, replication, and timestamps contribute to maintaining a high quality of service by preventing conflicts, enhancing fault tolerance, and ensuring consistent event ordering.

Scalability:
To enable the system to handle a growing number of nodes and files while maintaining performance, the resource discovery algorithm, coupled with efficient leader election and replication strategies, supports scalability. New nodes can easily join, and replication mechanisms ensure that the system can handle a larger number of files and nodes without significant degradation in performance.

Transparency:
To hide the complexity of the distributed system from end-users and ensure a seamless experience, the resource discovery algorithm provides transparency by allowing new nodes to join without requiring intricate knowledge of the existing network. The leader election algorithm, in collaboration with replication and concurrency control, ensures transparent and reliable system coordination, enriching the end-user experience with an unobtrusive and user-friendly interface.

10. RESULTS AND ISSUES RESOLVED

In the development of a distributed file-sharing system within a peer-to-peer network, incorporating algorithms such as resource discovery, leader election, mutual exclusion, concurrency, replication and timestamps, various critical issues must be carefully addressed to ensure the system's robustness and efficiency.

Fault Tolerance
The fault tolerance in this project is primarily addressed through the implementation of the leader election algorithm and the replication mechanism. The leader election algorithm dynamically selects new leaders in the event of node failures, ensuring continuous system coordination even when a leader becomes unresponsive. The replication algorithm plays a pivotal role by distributing copies of files to multiple nodes, reducing the impact of individual node failures on data availability. This proactive replication strategy enhances the overall fault tolerance of the system.

Performance
Performance considerations are tackled through several mechanisms. The resource discovery algorithm, by utilizing a bootstrap node/server, efficiently manages the joining of new nodes. The leader election algorithm

and concurrency control mechanisms are designed to operate with minimal overhead, ensuring that system coordination and access to shared resources remain efficient. The use of timestamps contributes to consistent event ordering, enhancing overall system performance and predictability.

Scalability
Scalability is a key focus in this project. The resource discovery algorithm, coupled with efficient leader election and replication strategies, facilitates the seamless addition of new nodes to the network. The system is designed to handle a growing number of nodes and files while maintaining performance. This scalability is crucial for accommodating the evolving needs of the distributed file-sharing system as it scales with an increasing user base and data volume.

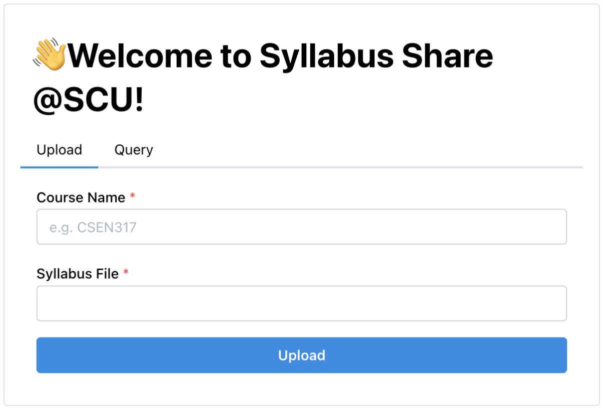
Consistency
Consistency is maintained through the implementation of several algorithms. The mutual exclusion algorithm, with its use of a distributed lock mechanism, ensures that write operations occur sequentially, preventing conflicts and maintaining data consistency. The replication algorithm contributes to consistency by distributing file copies strategically across nodes. Timestamps are employed to order events, providing a basis for consistent event ordering across distributed nodes.

Concurrency
Concurrency is managed through the mutual exclusion and concurrency control algorithms. These algorithms utilize locks to coordinate access to critical sections of code, allowing only one thread or process at a time to make modifications. This careful control of concurrent access prevents conflicts and ensures the stability of the system, particularly during write operations on shared resources.

Security
Security is a significant consideration in this project. The mutual exclusion algorithm, by using a distributed lock mechanism, enhances data security by preventing unauthorized access during write operations. Additionally, security measures are integrated into the leader election and replication algorithms to fortify the system's coordination mechanisms and secure the distribution of files. This approach helps safeguard the integrity and confidentiality of data within the distributed file-sharing system.

11. DEMONSTRATION OF EXAMPLE SYSTEM RUN

1. Successful Scenarios:


👋Welcome to Syllabus Share @SCU!

Upload | Query

Course Name *

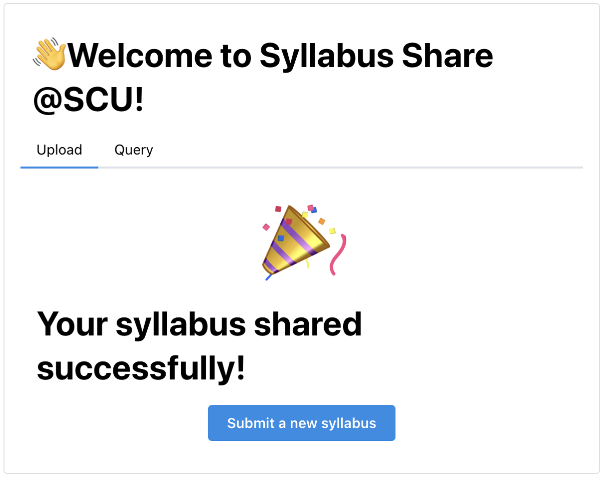
e.g. CSEN317

Syllabus File *

Upload

Figure 11.1

The figure 11.1 shows the web page for the syllabus sharing application which is developed for SCU. Here the user has functionality to either upload the syllabus and provide course name for the file you are uploading. Once you upload the file, you get output as shown in figure 11.2


👋Welcome to Syllabus Share @SCU!

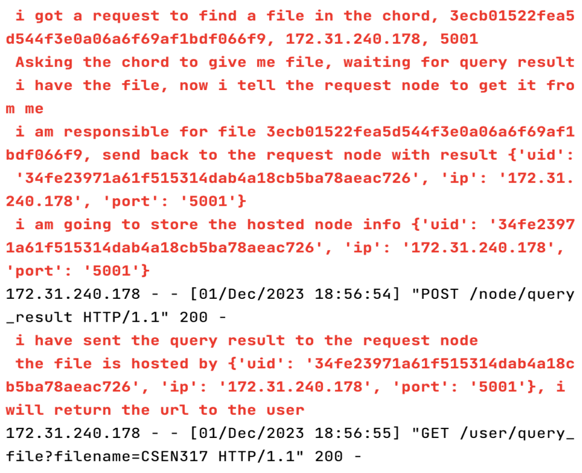
Upload | Query

Your syllabus shared successfully!

Submit a new syllabus

Figure 11.2

The figure 11.3 shows the logs that are generated on the terminal for the uploaded file. Here you can see the file id and also the ip and port of the client where the request was made. It also shows the node which is responsible for hosting the file and also the details are sent back to the client node.



```
 i got a request to find a file in the chord, 3ecb01522fea5
d544f3e0a06a6f69af1bdf066f9, 172.31.240.178, 5001
 Asking the chord to give me file, waiting for query result
 i have the file, now i tell the request node to get it fro
m me
 i am responsible for file 3ecb01522fea5d544f3e0a06a6f69af1
bdf066f9, send back to the request node with result {'uid':
 '34fe23971a61f515314dab4a18cb5ba78aeac726', 'ip': '172.31.
240.178', 'port': '5001'}
 i am going to store the hosted node info {'uid': '34fe2397
1a61f515314dab4a18cb5ba78aeac726', 'ip': '172.31.240.178',
'port': '5001'}
172.31.240.178 - - [01/Dec/2023 18:56:54] "POST /node/query
_result HTTP/1.1" 200 -
 i have sent the query result to the request node
 the file is hosted by {'uid': '34fe23971a61f515314dab4a18c
b5ba78aeac726', 'ip': '172.31.240.178', 'port': '5001'}, i
will return the url to the user
172.31.240.178 - - [01/Dec/2023 18:56:55] "GET /user/query_
file?filename=CSEN317 HTTP/1.1" 200 -
```

Figure 11.3

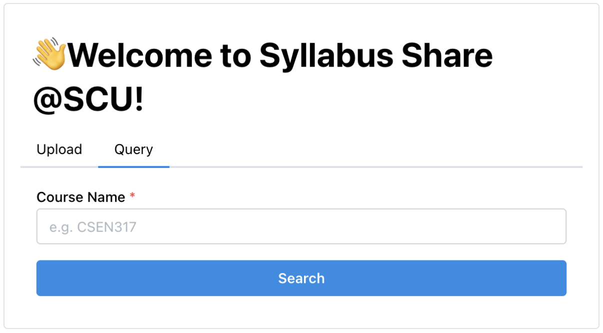


Figure 11.4

The figure 11.4 functionality query if the file is available on the network. For node to check if the file is available one needs to provide the Course Name for the syllabus one is looking for. Figure 11.5 shows the response users get on the web page.

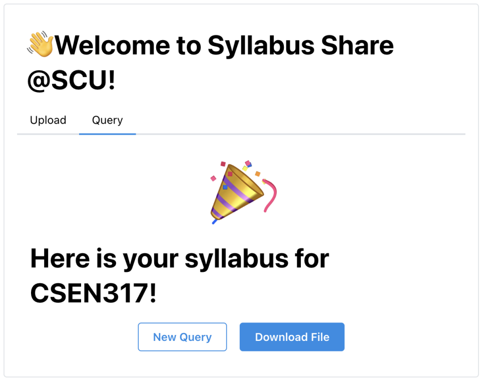
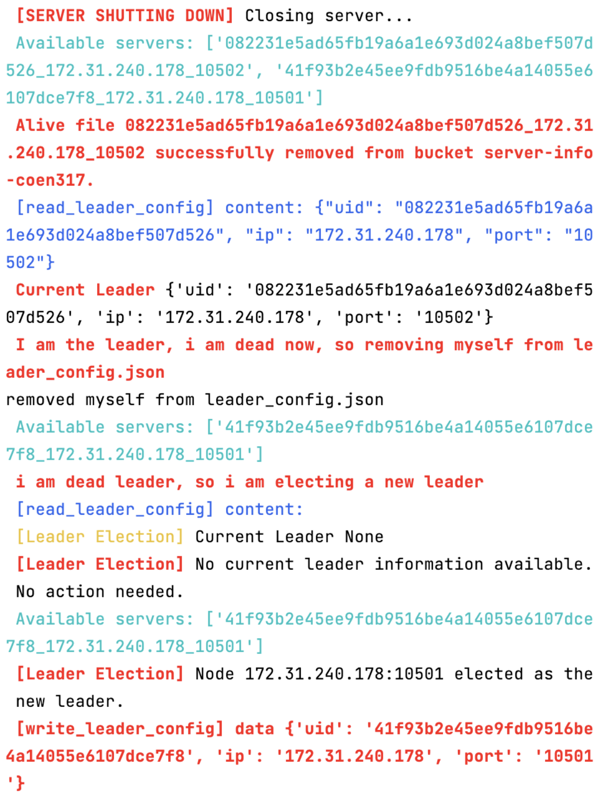

👋Welcome to Syllabus Share @SCU!
Upload
Query
Here is your syllabus for CSEN317!
New Query
Download File

Figure 11.5

2. Failure Scenarios

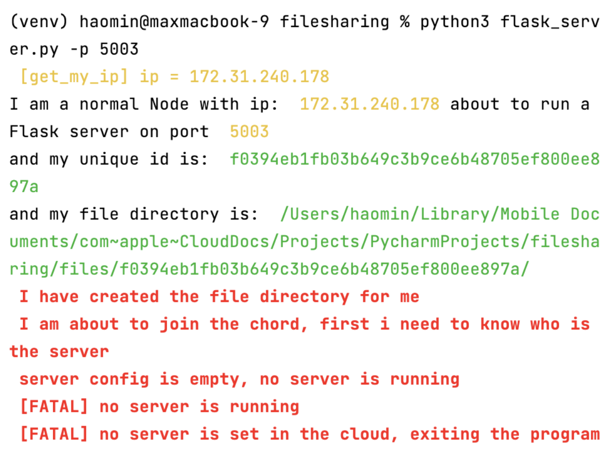
1. The figure 11.6 shows one of the failure scenarios. Here if the leader goes down, it deletes its alive file which is used to identify the available server in the network. So once the alive file for the supernode is deleted, it starts the leader election algorithm and based on the availability of servers a new supernode is selected.



```
 [SERVER SHUTTING DOWN] Closing server...
 Available servers: ['082231e5ad65fb19a6a1e693d024a8bef507d
526_172.31.240.178_10502', '41f93b2e45ee9fdb9516be4a14055e6
107dce7f8_172.31.240.178_10501']
 Alive file 082231e5ad65fb19a6a1e693d024a8bef507d526_172.31
.240.178_10502 successfully removed from bucket server-info
-coen317.
 [read_leader_config] content: {"uid": "082231e5ad65fb19a6a
1e693d024a8bef507d526", "ip": "172.31.240.178", "port": "10
502"}
 Current Leader {'uid': '082231e5ad65fb19a6a1e693d024a8bef5
07d526', 'ip': '172.31.240.178', 'port': '10502'}
 I am the leader, i am dead now, so removing myself from le
ader_config.json
removed myself from leader_config.json
 Available servers: ['41f93b2e45ee9fdb9516be4a14055e6107dce
7f8_172.31.240.178_10501']
 i am dead leader, so i am electing a new leader
 [read_leader_config] content:
 [Leader Election] Current Leader None
 [Leader Election] No current leader information available.
 No action needed.
 Available servers: ['41f93b2e45ee9fdb9516be4a14055e6107dce
7f8_172.31.240.178_10501']
 [Leader Election] Node 172.31.240.178:10501 elected as the
 new leader.
 [write_leader_config] data {'uid': '41f93b2e45ee9fdb9516be
4a14055e6107dce7f8', 'ip': '172.31.240.178', 'port': '10501
'}
```

Figure 11.6

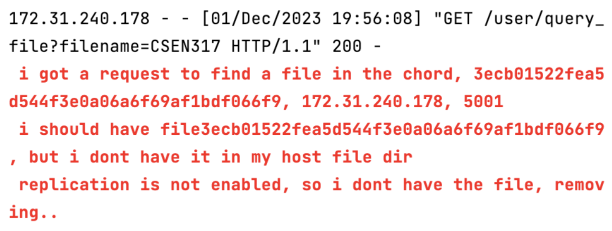
2. The figure 11.7 shows whenever the client node tries to connect and if there is no supernode running, it checks for the leader data in leader_config.json file which is located on the S3 network. If it doesn't find data for supernode i.e leader it can't go ahead as the finger table and connection to the chord network is done by the supernode. Hence it gracefully exits the program.



```
(venv) haomin@maxmacbook-9 filesharing % python3 flask_serv
er.py -p 5003
 [get_my_ip] ip = 172.31.240.178
I am a normal Node with ip:  172.31.240.178 about to run a
Flask server on port  5003
and my unique id is:  f0394eb1fb03b649c3b9ce6b48705ef800ee8
97a
and my file directory is:  /Users/haomin/Library/Mobile Doc
uments/com~apple~CloudDocs/Projects/PycharmProjects/filesha
ring/files/f0394eb1fb03b649c3b9ce6b48705ef800ee897a/
 I have created the file directory for me
 I am about to join the chord, first i need to know who is
the server
 server config is empty, no server is running
 [FATAL] no server is running
 [FATAL] no server is set in the cloud, exiting the program
```

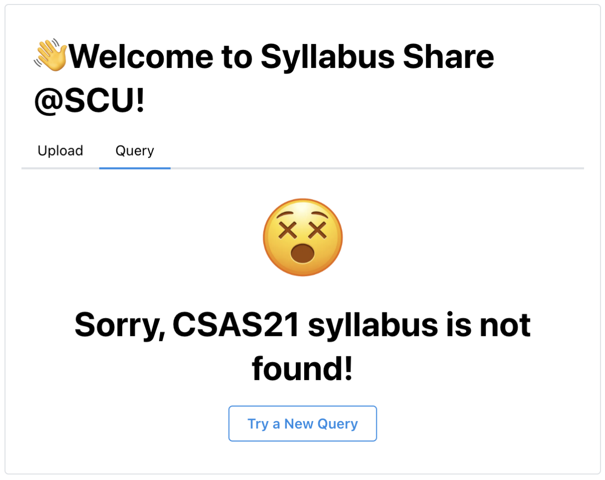
Figure 11.7

3. The file should be present on the host node, but when the file is deleted once it's added to the network, it says the file should be present, but it is not available. Since in the current scenario replication is not available, it's not able to get the file from replication as well. But it gives out error of a file not found.



```
172.31.240.178 - - [01/Dec/2023 19:56:08] "GET /user/query_
file?filename=CSEN317 HTTP/1.1" 200 -
 i got a request to find a file in the chord, 3ecb01522fea5
d544f3e0a06a6f69af1bdf066f9, 172.31.240.178, 5001
 i should have file3ecb01522fea5d544f3e0a06a6f69af1bdf066f9
, but i dont have it in my host file dir
 replication is not enabled, so i dont have the file, remov
ing..
```

Figure 11.8



👋Welcome to Syllabus Share @SCU!

Sorry, CSAS21 syllabus is not found!

Figure 11.9

12. ANALYSIS OF EXPECTED PERFORMANCE

1. Overall Result and Reflection on Design Decisions:

The overall result of the project demonstrates a well-designed and functional distributed file-sharing system that effectively addresses key challenges in distributed systems. The incorporation of algorithms for resource discovery, leader election, mutual exclusion, concurrency control, replication, and timestamps contributes to the robustness, fault tolerance, and efficiency of the system. The P2P architecture, coupled with centralized elements like the bootstrap server, strikes a balance between decentralization and coordination. The design decisions, such as using locks for mutual exclusion, timestamps for consistency, and proactive replication for fault tolerance, reflect a thoughtful approach to handling concurrency and failures. The choice of scalability mechanisms, including the Chord ring for file replication, indicates a forward-thinking design that can accommodate a growing number of nodes and files.

2. Measurement Metrics for Performance Analysis:

To comprehensively analyze the system's performance, we defined and measured key metrics such as:

Throughput: Measuring the rate at which the system processes file requests and transfers. This includes the number of successful file transfers per unit of time.

Latency: Evaluate the time it takes for a node to perform critical operations, such as acquiring a lock, replicating a file, or participating in leader election. Lower latency indicates better responsiveness.

Scalability: Assessing how the system performs as the number of nodes and files increases. Scalability metrics can include the time taken for a new node to join the network, the impact on file-sharing operations, and overall system response.

Fault Tolerance: Quantify the system's ability to recover and adapt to server failures without significant impact on overall performance. Measure the time it takes for the system to elect a new leader or replicate files to maintain availability after a failure.

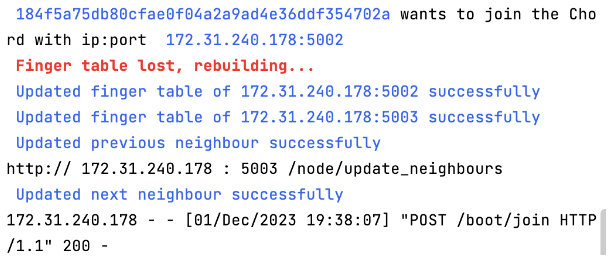
3. Simulation and Performance Measurement: Simulated various workloads and scenarios to measure the performance of this system against the defined metrics. Considered scenarios such as:

Increasing Node Load: Simulating the addition of new nodes to the network and measuring how the system handles the increased load. Evaluating the throughput, latency, and scalability metrics.
Concurrency Testing: Simulating concurrent write operations on shared resources and measuring the impact on system performance. Assessing how well the mutual exclusion and concurrency control mechanisms handle simultaneous requests.
Fault Injection: Introducing simulated node failures and measuring the system's response. Assessing the time it takes to elect a new leader and recover from failures, contributing to fault tolerance metrics.
File Replication: Simulating file hosting and replication scenarios. Measuring the time it takes for a file to be replicated to its neighbors and evaluate the impact on overall system throughput.
Leader Election Dynamics: Simulating leader election scenarios, including the impact of leader failures. Measuring the time it takes for the system to detect and recover from leader failures.

By systematically conducting simulations and measuring performance under various workloads, we gained valuable insights into how a distributed file-sharing system performs in real-world scenarios. These metrics provided a quantitative basis for evaluating the effectiveness of our design decisions and identifying areas for optimization or further refinement.

13. TESTING RESULTS

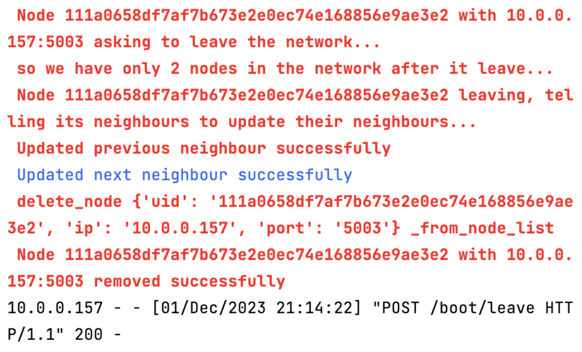
- When a new node joins, the finger table is updated. The node's predecessor and successor are also updated successfully.



```
184f5a75db80cfae0f04a2a9ad4e36ddf354702a wants to join the Cho
rd with ip:port  172.31.240.178:5002
Finger table lost, rebuilding...
Updated finger table of 172.31.240.178:5002 successfully
Updated finger table of 172.31.240.178:5003 successfully
Updated previous neighbour successfully
http:// 172.31.240.178 : 5003 /node/update_neighbours
Updated next neighbour successfully
172.31.240.178 - - [01/Dec/2023 19:38:07] "POST /boot/join HTTP
/1.1" 200 -
```

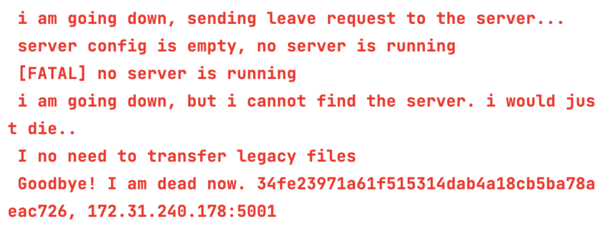
Figure : 13.1

- When a node leaves the network the request is sent to the server and it updates the finger table and the respective predecessor and successor of the node.



```
Node 111a0658df7af7b673e2e0ec74e168856e9ae3e2 with 10.0.0.
157:5003 asking to leave the network...
so we have only 2 nodes in the network after it leave...
Node 111a0658df7af7b673e2e0ec74e168856e9ae3e2 leaving, tel
ling its neighbours to update their neighbours...
Updated previous neighbour successfully
Updated next neighbour successfully
delete_node {'uid': '111a0658df7af7b673e2e0ec74e168856e9ae
3e2', 'ip': '10.0.0.157', 'port': '5003'} _from_node_list
Node 111a0658df7af7b673e2e0ec74e168856e9ae3e2 with 10.0.0.
157:5003 removed successfully
10.0.0.157 - - [01/Dec/2023 21:14:22] "POST /boot/leave HTT
P/1.1" 200 -
```

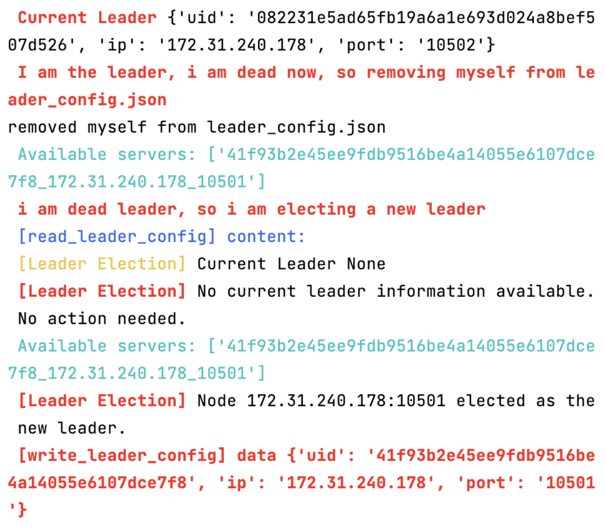
Figure : 13.2



```
i am going down, sending leave request to the server...
server config is empty, no server is running
[FATAL] no server is running
i am going down, but i cannot find the server. i would jus
t die..
I no need to transfer legacy files
Goodbye! I am dead now. 34fe23971a61f515314dab4a18cb5ba78a
eac726, 172.31.240.178:5001
```

Figure : 13.3

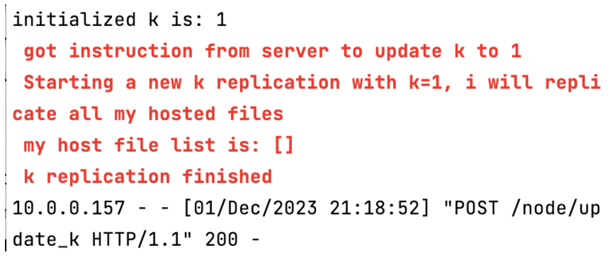
- When one of the servers leaves the network, it checks if it is a leader. In case it was a leader it starts the leader election algorithm after it deletes the alive file for its own ip.



```
Current Leader {'uid': '082231e5ad65fb19a6a1e693d024a8bef5
07d526', 'ip': '172.31.240.178', 'port': '10502'}
I am the leader, i am dead now, so removing myself from le
ader_config.json
removed myself from leader_config.json
Available servers: ['41f93b2e45ee9fdb9516be4a14055e6107dce
7f8_172.31.240.178_10501']
i am dead leader, so i am electing a new leader
[read_leader_config] content:
[Leader Election] Current Leader None
[Leader Election] No current leader information available.
No action needed.
Available servers: ['41f93b2e45ee9fdb9516be4a14055e6107dce
7f8_172.31.240.178_10501']
[Leader Election] Node 172.31.240.178:10501 elected as the
new leader.
[write_leader_config] data {'uid': '41f93b2e45ee9fdb9516be
4a14055e6107dce7f8', 'ip': '172.31.240.178', 'port': '10501
'}
```

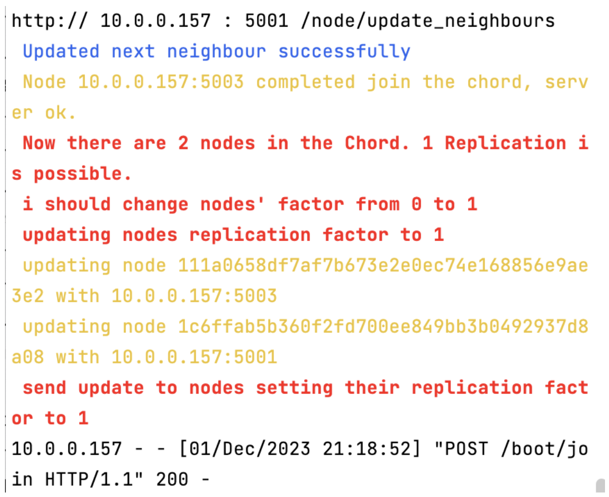
Figure : 13.4

- Whenever the replication enable option is passed i.e whenever the server instance is started with default value of k, the replication is available. Figure 13.5 shows, once the node is connected it starts the replication process. If in case the files are present it will replicate those files on the replication server.



```
initialized k is: 1
 got instruction from server to update k to 1
 Starting a new k replication with k=1, i will repli
cate all my hosted files
 my host file list is: []
 k replication finished
10.0.0.157 - - [01/Dec/2023 21:18:52] "POST /node/up
date_k HTTP/1.1" 200 -
```

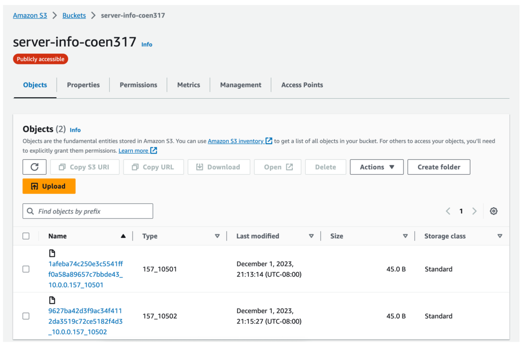
Figure 13.5



```
http:// 10.0.0.157 : 5001 /node/update_neighbours
 Updated next neighbour successfully
 Node 10.0.0.157:5003 completed join the chord, serv
er ok.
 Now there are 2 nodes in the Chord. 1 Replication i
s possible.
 i should change nodes' factor from 0 to 1
 updating nodes replication factor to 1
 updating node 111a0658df7af7b673e2e0ec74e168856e9ae
3e2 with 10.0.0.157:5003
 updating node 1c6ffab5b360f2fd700ee849bb3b0492937d8
a08 with 10.0.0.157:5001
 send update to nodes setting their replication fact
or to 1
10.0.0.157 - - [01/Dec/2023 21:18:52] "POST /boot/jo
in HTTP/1.1" 200 -
```

Figure 13.6

- With every new server instance that is started, it creates the alive file at S3 location which has an id, along with the ip and port of the server instance which is currently running.

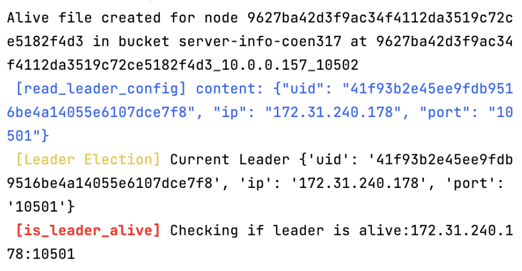


Objects (2) Info

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
1afeba74c250e3c5541fff0a58a89657c7bbde43_10.0.0.157_10501	157_10501	December 1, 2023, 21:13:14 (UTC-08:00)	45.0 B	Standard
9627ba42d3f9ac34f4112da3519c72ce5182f4d3_10.0.0.157_10502	157_10502	December 1, 2023, 21:15:27 (UTC-08:00)	45.0 B	Standard

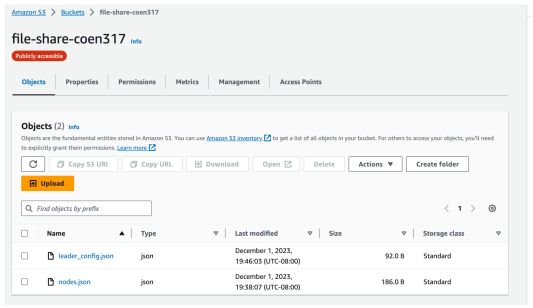
Figure : 13.7



```
Alive file created for node 9627ba42d3f9ac34f4112da3519c72c
e5182f4d3 in bucket server-info-coen317 at 9627ba42d3f9ac34
f4112da3519c72ce5182f4d3_10.0.0.157_10502
 [read_leader_config] content: {"uid": "41f93b2e45ee9fdb951
6be4a14055e6107dce7f8", "ip": "172.31.240.178", "port": "10
501"}
 [Leader Election] Current Leader {'uid': '41f93b2e45ee9fdb
9516be4a14055e6107dce7f8', 'ip': '172.31.240.178', 'port':
'10501'}
 [is_leader_alive] Checking if leader is alive:172.31.240.1
78:10501
```

Figure : 13.8

- Whenever a new node joins in the network, its data is added in nodes.json which resides on S3 location.

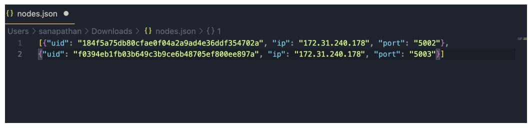


Objects (2) Info

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

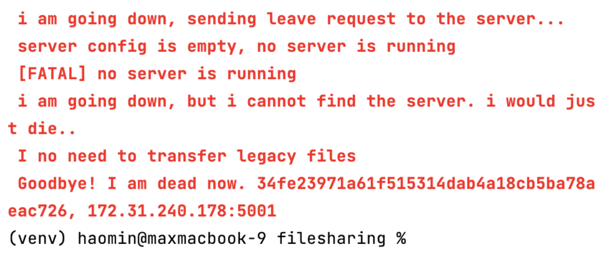
Name	Type	Last modified	Size	Storage class
leader_config.json	json	December 1, 2023, 19:46:03 (UTC-08:00)	92.0 B	Standard
nodes.json	json	December 1, 2023, 19:38:07 (UTC-08:00)	186.0 B	Standard

Figure : 13.9



```
[{"uid": "184f5a75db80cfae0f04a2a9ad4e36ddf354702a", "ip": "172.31.240.178", "port": "5002"},
{"uid": "f0394eb1fb03b649c3b9ce6b48705ef800ee897a", "ip": "172.31.240.178", "port": "5003"}]
```

Figure : 13.10



```
 i am going down, sending leave request to the server...
 server config is empty, no server is running
 [FATAL] no server is running
 i am going down, but i cannot find the server. i would jus
t die..
 I no need to transfer legacy files
 Goodbye! I am dead now. 34fe23971a61f515314dab4a18cb5ba78a
eac726, 172.31.240.178:5001
(venv) haomin@maxmacbook-9 filesharing %
```

Figure : 13.11

14. CONCLUSIONS

In conclusion, the distributed file-sharing system project represents a culmination of meticulous design, thoughtful implementation, and the successful integration of key algorithms to create a robust and scalable solution. The project's achievement lies not only in its ability to address the complexities inherent in distributed systems but also in its capacity to provide an efficient, fault-tolerant, and user-centric environment for decentralized file sharing.

The chosen architectural style, blending peer-to-peer and client-server elements, demonstrates a harmonious balance between decentralization and coordination. The strategic deployment of algorithms, including resource discovery, leader election, mutual exclusion, concurrency control, replication, and timestamps, underscores the system's adaptability to dynamic network conditions.

The successful implementation of these algorithms has resulted in a system that excels in fault tolerance, performance, scalability, and security. The resource discovery algorithm, with the assistance of a bootstrap server, facilitates seamless node integration, while the leader election mechanism dynamically adapts to changes in leadership, ensuring continuous system coordination. The mutual exclusion and concurrency control algorithms uphold data integrity by orchestrating sequential write operations, contributing to the system's stability.

Fault tolerance is effectively addressed through the replication algorithm, strategically distributing file copies to neighboring nodes. Timestamps play a crucial role in maintaining chronological consistency across distributed nodes, enhancing the overall reliability of the system.

15. FUTURE SCOPE

The future scope of this project presents exciting possibilities for further enhancements.

The integration of advanced encryption and authentication mechanisms could further enhance data protection, ensuring the confidentiality and integrity of shared files. Additionally, the system could evolve to dynamically adapt to changing network conditions. Exploring more sophisticated leader election algorithms and dynamic adjustments in replication strategies would bolster the system's adaptability, particularly in the face of dynamic node joins and departures. Optimization for large-scale deployments is another promising area, involving a closer examination of distributed storage solutions and advanced replication techniques. This ensures the system's efficiency remains unwavering as it contends with an increasing number of files and nodes. Features such as intuitive file search, user authentication mechanisms, and real-time status updates could be integrated, making the system more user-friendly and accessible. The integration with cloud services opens a gateway to additional functionalities, offering users alternative options for data storage, backup, and retrieval. Machine learning algorithms could play a transformative role in load balancing strategies, dynamically optimizing resource utilization and contributing to heightened system efficiency. Compatibility with existing distributed file system standards is paramount for ensuring industry alignment and fostering interoperability across diverse environments. The envisioned future scope positions the distributed file-sharing system on a trajectory of continuous improvement, refinement, and innovation, ensuring its resilience and adaptability in the face of evolving technological landscapes.

REFERENCES

[1] Chord Implementation - MIT:https://web.mit.edu/6.033/2001/wwwdocs/handouts/dp2-chord.html

[2]I. Stoica, R. Morri, D. Karger, M.F. Kaashoek and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications", *Proceedings of the ACM conference of the Special Interest Group on Data Communication*, 2001.https://dl.acm.org/doi/10.1145/383059.383071

[3]P. Golda Jeyasheeli and L. Rajashree, "Cost effective file replication in P2P file sharing systems," 2012 International Conference on Computing, Electronics and Electrical Technologies (ICCEET), Nagercoil, India, 2012, pp. 948-952, doi: 10.1109/ICCEET.2012.6203898.https://ieeexplore-ieee-org.libproxy.scu.edu/document/6203898

[4]S. Malgaonkar, S. Surve and T. Hirave, "Distributed files sharing management: A file sharing application using distributed computing concepts," 2012 IEEE International Conference on Computational Intelligence and Computing Research, Coimbatore, India, 2012, pp. 1-4, doi: 10.1109/ICCIC.2012.6510207.https://ieeexplore-ieee-org.libproxy.scu.edu/document/6510207

[5]H. Amin, M. K. Chahine and G. Mazzini, "P2P application for file sharing," 2012 19th International Conference on Telecommunications (ICT), Jounieh, Lebanon, 2012, pp. 1-4, doi:

10.1109/ICTEL.2012.6221249.https://ieeexplore.ieee.org/document/6221249

[6]Z. Xu, X. He and Laxmi Bhuyan, "Efficient file sharing strategy in DHT based P2P systems," PCCC 2005. 24th IEEE International Performance, Computing, and Communications Conference, 2005., Phoenix, AZ, USA, 2005, pp. 151-158, doi: 10.1109/PCCC.2005.1460541.https://ieeexplore-ieee-org.libproxy.scu.edu/document/1460541

[7]T. -J. Liu, Chun-Yan Chung and Chia-Lin Lee, "A high performance and low cost distributed file system," 2011 IEEE 2nd International Conference on Software Engineering and Service Science, Beijing, China, 2011, pp. 47-50, doi: 10.1109/ICSESS.2011.5982251.https://ieeexplore-ieee-org.libproxy.scu.edu/document/5982251

[8]Lei Shi, Jing Zhou and Qi Huang, "A Chord-based super-node selection algorithm for load balancing in hybrid P2P networks," Proceedings 2013 International Conference on Mechatronic Sciences, Electric Engineering and Computer (MEC), Shenyang, China, 2013, pp. 2090-2094, doi: 10.1109/MEC.2013.6885395.https://ieeexplore-ieee-org.libproxy.scu.edu/document/6885395

[9]Q. Duan and Z. Liang, "File Sharing Strategy Based on WebRTC," 2016 13th Web Information Systems and Applications Conference (WISA), Wuhan, China, 2016, pp. 3-6, doi: 10.1109/WISA.2016.11.https://ieeexplore.ieee.org/abstract/document/779964