A Distributed File System

1st Hanuma Sashank Samudrala
Dept. of Computer Science
University of Maryland Baltimore County
Baltimore, USA
hanumas1@umbc.edu

2nd Ashish Athimamula
Dept. of Computer Science
University of Maryland Baltimore County
Baltimore, USA
ka42455@umbc.edu

***Abstract*—In a computer system, sharing resources becomes important either for economic reasons or because some applications require it. When it's necessary to share devices and their data among multiple users or computers for extended periods, Distributed File Systems (DFS) come into play. These systems enable users on different computers, even if they're physically far apart, to share data and storage resources using a shared file system. Essentially, a distributed file system acts like a shared space where multiple users or computers can access and manage files as if they're stored locally, regardless of where the files physically reside. We have come up with a solution that addresses the challenges for the problem of having a common file system, providing flexibility, reliability, availability, and efficiency for managing large data volumes. Our approach makes our system modular by distributing three tasks to three servers namely active server, DB server, lock service server, and other servers to interact with the clients.**

***Index Terms*—Distributed File Systems(DFS), client, server**

I. Introduction

Distributed File Systems (DFS) have gained significant attention in contemporary research. This technology facilitates remote file storage and access, mirroring the operations used for local files. Users can access these files from any network-connected computer, offering seamless accessibility. Before the evolution of computer networks in the 1980s, file sharing relied on a method called "sneakernet." This involved copying files onto floppy disks, physically transporting them to another computer, and duplicating the data. However, with the evolution of networks, it became apparent that traditional file systems had numerous limitations, rendering them inadequate for multiuser environments.

The seamless and concurrent sharing of files across multiple servers in a networked environment poses a significant challenge. Ensuring data consistency, accessibility, and reliability while accommodating various file operations is pivotal in distributed systems. The importance lies in establishing a robust infrastructure that enables fault-tolerant file handling, minimizes downtime, and ensures data integrity, particularly in scenarios where multiple clients interact with distributed servers.

Our distributed file system addresses this challenge by implementing a comprehensive solution for file operations across interconnected servers. Leveraging a client-server architecture, it facilitates the creation, reading, writing, and deletion of files while maintaining consistency and reliability. The system utilizes primary and replicated copies, enabling seamless access to files across servers. Through locking mechanisms and systematic replication strategies, the solution guarantees orderly file access, synchronization, and fault tolerance across the distributed network.

II. Description

In our system, the users are the clients connecting to the servers for making various file operations like create, delete, open, close, read, write, seek on each file.

Create: The "create" operation generates a new file in the file system. It involves allocating storage space, setting initial attributes, and establishing metadata for the new file. In a distributed system, this operation include coordinating among servers to create a file, ensuring consistency across replicas, and updating directory information.

Delete: Deletes a specified file from the file system, removing the file entry from the directory structure and frees up the allocated storage space. In a distributed setup, deleting a file might involve ensuring that all replicas and primary copy of the file are removed to maintain consistency across servers.

Open: "Open" grants access to a file for reading, it establishes a connection or session between the user or program and the file by displaying it's content. In a distributed system, this operation is responsible for locating the file across distributed servers, identifying the appropriate replica or primary copy based on the requested operation, acquiring locks or establishing communication channels with the server holding the file to facilitate subsequent read, which ensures resource allocation and management for the duration of the open session.

Close: "Close" terminates the access to a file, ending the connection or session. Finalizes the access to the file, releasing resources and terminating the connection or session established during the open operation. In a distributed system, it requires communication with the relevant server to release any locks or resources associated with the file, finalizing any ongoing transactions or operations related to the file and ensuring data integrity by facilitating the completion of the file access session, allowing other users or processes to access the file afterward.

Read: "Read" retrieves data from a file stored in the file system. In a distributed system, it requires locating the file across servers, potentially on different nodes in a network and also requires communication with the relevant server to release any locks or resources associated with the file. Ensures

data consistency and accuracy by accessing the correct replica of the file. And involves communication with the appropriate server holding the file or its replica to retrieve the requested data.

Write: "Write" stores data into a file in the file system. Involves adding new data or modifying existing content within a file. The operation appends new data at the end of the existing file. In a distributed system, it is responsible for locating the file across servers, ensuring write operations occur on the appropriate replica to maintain data consistency. Involves coordinating simultaneous writes by multiple clients or processes by managing locks or employing concurrency control mechanisms. Ensures that data is accurately propagated across replicas to maintain coherence and reliability within the distributed system. Involves replication strategies to update all replicas after a successful write operation.

Seek: "Seek" repositions the file pointer within a file. It moves the file pointer to a specified position, allowing subsequent reads to occur from that point to the end of the file. This operation has nothing to do with replication.

In addition to these normal file operations, the client can get the list of files available in the system by an option called list of files in the system. This command displays the list of all available files created by various clients over different servers.

Primary copy: "Primary copy" holds pivotal importance, representing the initial version of a file created or modified by a client connected to a server. This primary copy resides within the directory managed by the connected server, establishing immediate ownership and access for the client. Integral to this system is the DB Server, serving as a central repository that meticulously maintains crucial information about primary and replica copies distributed across servers. It tracks details such as file names, locations of primary copies, and replicas across the network. This primary copy acts as the authoritative source for modifications, ensuring a controlled process for updates and facilitating synchronization across replica copies. Changes to the primary copy trigger synchronization mechanisms that propagate these modifications across replicas, ensuring uniformity and coherence of data.

Server Replication: In this distributed file system, file replication operates distinctly during file creation and write operations.

When Create: In a distributed file system, file creation involves meticulous replication strategies to ensure fault tolerance and data redundancy. When a file is created, the overseeing server employs a process facilitated by the DB Server to identify servers devoid of replicas for the new file. Subsequently, one of these identified servers is randomly selected to host a replica, optimizing the distribution of file replicas across the network. This systematic approach ensures an efficient replication scheme, allowing for robust tracking and management of file copies, thereby enhancing fault tolerance and bolstering data redundancy within the distributed system.

When Write: During write operations in the system, the responsible server orchestrates a comprehensive synchronization process. Initially, it connects to the DB Server to access critical information regarding the primary copy of the file. Upon obtaining the necessary details, the server proceeds to update the primary copy with the intended modifications or new content. Subsequent to updating the primary copy, the server leverages information retrieved from the DB Server about the existing replicas distributed across various servers. Utilizing this data, the server systematically synchronizes the modifications made to the primary copy across all existing replicas. This meticulous synchronization process, managed by the DB Server's centralized control, ensures fault tolerance, robust data redundancy, and systematic synchronization of file content across the distributed environment. Ultimately, the DB Server plays a pivotal role in managing replication, guaranteeing a consistent and reliable file system across the distributed network.

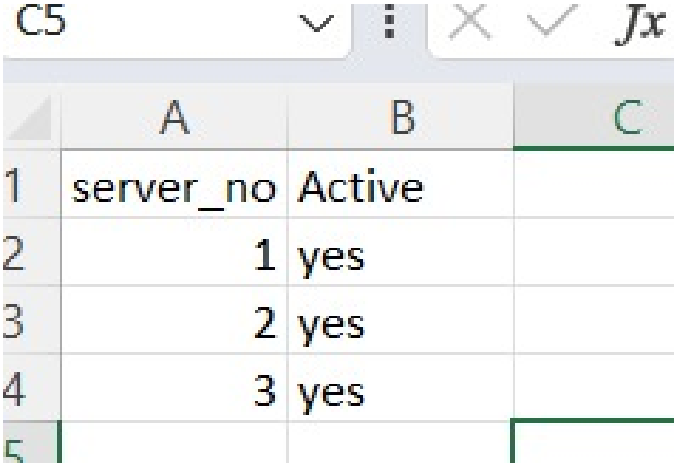
Locking Mechanism: In this distributed file system, exclusive write access to a file is governed by a fair locking mechanism. When Client 1 requests to write to a file, it locks the file, allowing reading but halting other write requests. Subsequent clients, like Client 2 or Client 3, waiting to write to the locked file, employ a polling mechanism with a 10-second timeout to check for an unlocked state. Upon unlocking, clients gain access in a First-In-First-Out (FIFO) manner: Client 2 writes first, followed by Client 3, ensuring fairness and orderly access. This approach maintains file integrity, preventing conflicts among write requests and granting write permissions sequentially based on the order of requests, fostering equitable file access in the distributed system.

Active Server: "Active Server" orchestrates vital communication and facilitates client-server interactions. When a server is operational, it continuously communicates its status to the Active Server using heartbeat messages. These messages serve as regular updates, indicating the server's availability and operational status. Concurrently, the Active Server maintains a comprehensive and up-to-date list of currently running servers within the distributed network.

When a client initiates a connection request to the system, the Active Server assumes responsibility for managing this interaction. It actively serves as a mediator, leveraging its knowledge of the running servers to guide the client to a suitable server for connection. This is achieved by the Active Server providing the address and port numbers of an available, responsive, and running server from its maintained list. This information empowers the client to establish a direct connection with the designated server specified by the Active Server's response.

This orchestration by the Active Server streamlines the client's connection process, effectively functioning as a directory or coordinator, guiding incoming clients to an operational server. By dynamically providing information on available servers, the Active Server optimizes resource utilization, enhances system reliability, and ensures efficient communication channels between clients and servers in the distributed environment.

Db Server: "DB Server" serves as a crucial repository,



	A	B	C
1	server_no	Active	
2	1	yes	
3	2	yes	
4	3	yes	

Fig. 1. Entries of active servers

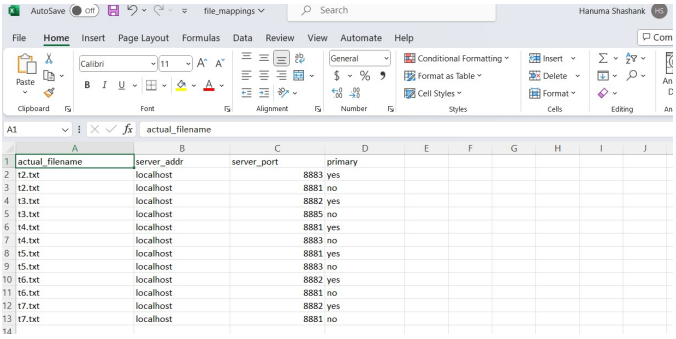
storing essential metadata and orchestrating file-related operations across multiple servers. When a file is created by a server within the network, the DB Server records it as the ”primary” copy, noting its server address for identification and management purposes. Additionally, during the replication process, upon the creation of file replicas across various servers, the DB Server diligently maintains records of these replicas, storing their respective addresses and port numbers.

In instances where a server lacks a specific file required for operations, it initiates a connection with the DB Server. This connection facilitates the retrieval of necessary file-related information and operations. During write operations, when the server intends to modify or update a file, the DB Server provides the address of the primary copy, enabling the server to access and make alterations to the authoritative version of the file.

Conversely, during read operations, when the server seeks to retrieve file data without intending to modify it, the DB Server provides the address of one of the replicated copies rather than the primary copy. This approach aims to optimize file access by directing read requests to one of the distributed replicas, distributing the load and enhancing performance across the network.

In essence, the DB Server acts as a centralized metadata repository, effectively managing file metadata, addressing, and facilitating access for servers within the distributed system. Its role in guiding servers to access primary or replicated copies based on the nature of the operation significantly contributes to system efficiency, load distribution, and optimized file access across the distributed environment.

Lock Server: ”Lock Server” within a distributed file system plays a pivotal role in managing exclusive access to files among multiple clients, ensuring orderly write operations and avoiding write conflicts. When a client intends to modify a file (Client 1 in this scenario), it requests a lock specifically for writing to that file. Upon receiving the lock, Client 1 gains exclusive write access, permitting modifications while allowing concurrent read operations by any client.



	actual_filename	server_addr	server_port	primary
2	t2.txt	localhost	8883	yes
3	t2.txt	localhost	8881	no
4	t3.txt	localhost	8882	yes
5	t3.txt	localhost	8885	no
6	t4.txt	localhost	8881	yes
7	t4.txt	localhost	8883	no
8	t5.txt	localhost	8881	yes
9	t5.txt	localhost	8883	no
10	t6.txt	localhost	8882	yes
11	t6.txt	localhost	8881	no
12	t7.txt	localhost	8882	yes
13	t7.txt	localhost	8881	no

Fig. 2. Db server maintaining file records

However, in cases where another client (like Client 2, Client 3, etc.) seeks write access to the same locked file, they must wait until the file is unlocked by the current writer (Client 1). The Lock Server handles this by maintaining a queue of write requests, ensuring fairness in access. Client 2, being the next in line, initiates a polling mechanism with a 10-second timeout to periodically check for the file’s unlock status. Once the file becomes available (unlocked) after Client 1 finishes writing, Client 2, the first in the queue, receives the lock and can commence writing.

This process continues in a sequential manner, honoring the First-In-First-Out (FIFO) principle. Client 3 gains access after Client 2 completes its write operation, followed by Client 4, and so on. The Lock Server effectively manages and grants access to clients requesting write permissions, fostering fairness and ensuring orderly write operations in the distributed file system. Its role as a coordinator for file locks facilitates a systematic and equitable approach to write access, preventing write conflicts and maintaining a consistent, orderly system for clients accessing files.

Characteristics of our proposed system:

Scalability: In our approach scalability revolves around the system’s ability to seamlessly expand and handle increasing demands without sacrificing performance or reliability. It involves the system’s capacity to accommodate additional servers, clients, and data volumes while maintaining operational efficiency. The system’s scalability is crucial in handling more concurrent write requests, accommodating growing data storage needs, and effectively managing increased communication and synchronization among servers. By implementing horizontal scaling, load balancing mechanisms, and robust coordination strategies, your distributed file system aims to ensure that it can efficiently grow, adapt, and manage escalated workloads without compromising on responsiveness or data consistency.

Location Transparency: Users interact with files without needing to know their physical location or which server holds the data. The system abstracts the file’s location, allowing users to access files uniformly regardless of where they are stored.

Concurrency Transparency: We achieved concurrency transparency when clients request file locks for writing without

needing to understand the intricate locking process among multiple clients. The system manages the locking mechanism transparently, ensuring fair access and orderly write operations.

Replication Transparency: The users in our system interact with files without being aware of the replicated copies or their locations. The system handles replication transparently, maintaining consistency and fault tolerance across replicas.

Performance Transparency: The modular approach of having three different servers eases the file server's load allowing users experience consistent performance regardless of varying server loads. The system transparently distributes client requests among servers, ensuring efficient resource utilization.

Availability: We designed our system in such a way that by creating a primary copy along with a replica on a separate server, we ensure uninterrupted service even if the server hosting the primary copy encounters issues. This approach enables seamless failover, allowing clients to access files from available replicas, thereby minimizing downtime and ensuring consistent data accessibility and reliability, even in the face of server disruptions or maintenance activities. We created randomly a single replica for a primary copy in any on of the server.

III. Architecture

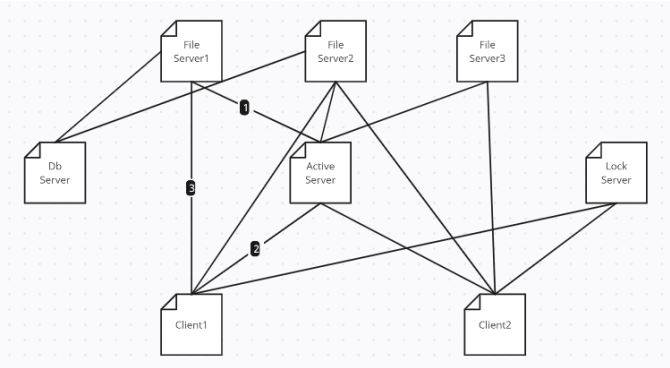


Fig. 3. Architecture diagram

Initially, DB server, Active server, and Lock servers are activated, forming the system's backbone, ensuring database management, server connectivity, and lock control. File servers are powered on to facilitate file storage and operations within the distributed system, enabling data handling across multiple nodes. File servers broadcast an active message to the Active server, signaling their readiness and availability for client connections and operations. Clients connect to the Active server, which directs them to a connected file server, establishing a connection for subsequent file operations. Clients communicate file system operations with the connected file server, initiating various file-handling tasks within the distributed environment. Upon file creation request, the primary copy resides in the connected file server, serving as the basis for subsequent operations on that file. Immediately after creating the primary copy, the file server replicates the file to another active server, ensuring redundancy and fault tolerance. File write operations occur exclusively on the primary copy file server, followed by immediate updates to all replicas for consistency. If the connected server lacks the primary copy, it requests the DB server for the primary copy server's address to enable write operations. Any modifications made on the primary copy server are promptly synchronized with corresponding replicas, maintaining uniformity across the system. For file reads unavailable on the connected server, clients prompt the DB server to access the primary copy server, retrieving and displaying the requested data. Clients request file locks from the lock server before writing to a file, ensuring exclusive access, and write permission only upon successful lock acquisition. Prior to file reads, clients verify the lock status on the lock server, allowing read access only when the file is not locked, ensuring controlled access. Implementing a fair lock system where multiple clients requesting write access follow a FIFO queue system, ensuring orderly access and fairness in write permissions. Clients seeking the system's file list contact the DB server, retrieving and displaying the comprehensive list of available files within the distributed system.

IV. Realization

: Python was utilized as the primary programming language for developing the distributed file system. Packages:

We have used the following packages in our system

socket: Facilitated network communication between servers and clients.

time: Assisted in managing time-related operations or delays within the system.

csv: Aided in handling CSV files or data parsing if required for specific functionalities.

os: Supported various operating system-related operations like file handling and directory management.

shutil: Enabled file operations such as copying, moving, and deleting files and directories.

random: Utilized for generating random values, possibly for replication or server selection processes.

collections: Supported specialized data structures, enhancing data handling or organization if needed.

The system was designed and implemented to run on diverse platforms supporting Python, ensuring platform agnosticism. The system relied on standard Python libraries, ensuring minimal external dependencies, thus enhancing portability and ease of deployment.

V. Performance Evaluation and Future Work

:

In this section, we present measurements that reflect on the design and implementation of the (DFS) prototype. Our discussion focuses on following questions:

How does the system handle file creation and replication across distributed servers?

How does the system enable file reading from a server that doesn't host the requested file (primary or replica)?

How is file writing facilitated on the primary copy and immediate updating in replicas from a server that doesn't host the file?

Process of file deletion, including both primary and replica file removal.

Does the system generate and provide a comprehensive list of available files within the distributed environment?

File locking mechanism, ensuring client access in a First-In-First-Out (FIFO) order.

1. Creation and Replication: The system initiates file creation in a connected server, establishing the primary copy. Simultaneously, the file is replicated in another active server, ensuring redundancy and fault tolerance.

2. Reading from Non-Hosting Server: For file reads unavailable on the connected server, the system contacts the Database (DB) server, which accesses the primary copy server to retrieve and display requested file data to the client.

3. Writing from Non-Hosting Server: In scenarios where the connected server lacks the primary copy, the server requests the primary copy server's address from the DB server, enabling write operations on the file.

4. File Deletion: File deletion triggers removal from both the hosting server (primary or replica) and the replicated copies across the distributed servers, ensuring complete removal.

5. List of Files: The system contacts the DB server, retrieves a comprehensive list of available files within the distributed system, and presents it to the client upon request. Using the sockets only.

6. File Locking and Access Order: File locking is managed through a dedicated lock server, enabling clients to acquire locks before write operations, ensuring orderly access in a FIFO manner for subsequent clients.

The **scalability analysis** conducted on our system involved augmenting the number of clients and servers to evaluate its performance. We systematically increased the workload by adding more clients and servers to the system, subsequently observing and recording the system's response time and throughput metrics. These metrics were meticulously tracked and documented as part of the scalability analysis.

We Observed gradual increment in response time with respect to no.of clients and gradual decreament in response time with increasing of servers(though seen some anomalies)

We have tested the system by increasing the workload and noted the performance of the system. We observed that on the whole, as we read or write larger amounts of data the response time and throughput of the system has reduced.

This process was graphically represented to illustrate the relationship between the increasing number of clients and servers against the corresponding response time and throughput metrics. The graphical representation allowed us to visualize the system's behavior under varying workloads, offering insights into its performance under increased load. This analysis provided valuable insights into the system's capacity to handle increased loads, aiding in identifying potential performance bottlenecks and optimizing the system for enhanced scalability and efficiency.

In the next section we will discuss about the scope and future work of our system.

VI. Future Work

: We are exploring the implementation of a backup server sys- tem to enhance node failure and disconnection management. This system aims to automatically redirect connections to corresponding backup nodes when a primary node is switched off. The envisioned setup includes the creation of backup servers for each node, providing a seamless transition for connecting clients and ensuring continuous system availability even in the face of node failures.

As of now we have included backup database for every fileserver to store the current status of the fileserver before turning off and retrieving back the previous state information of the server after turning on.

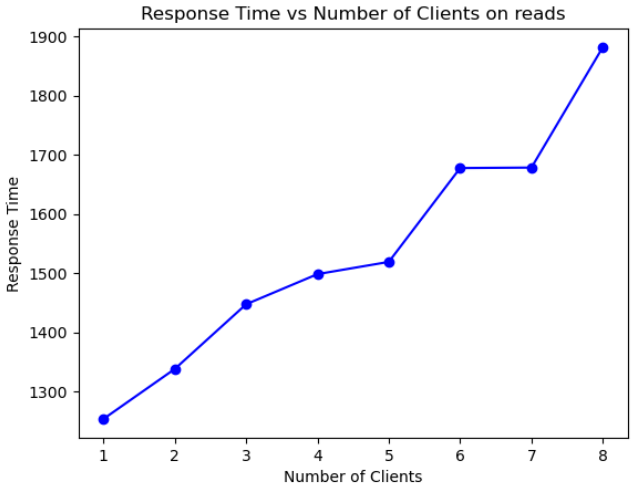


Fig. 4. Clients vs Response time on reads

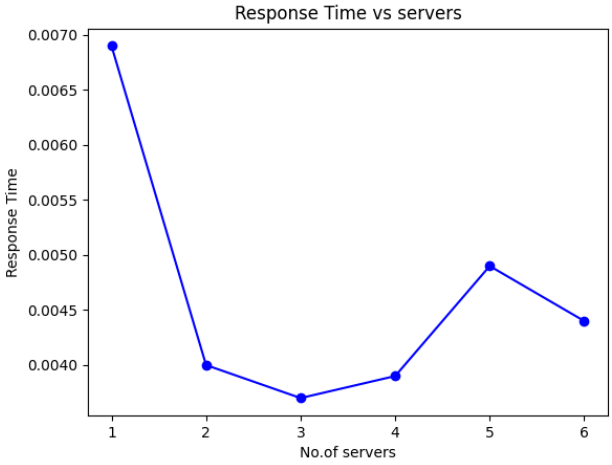


Fig. 5. Servers vs Response time on reads

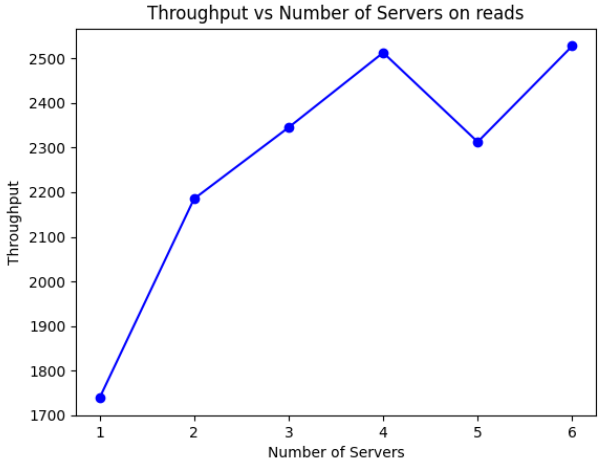


Fig. 6. Servers vs Throughput

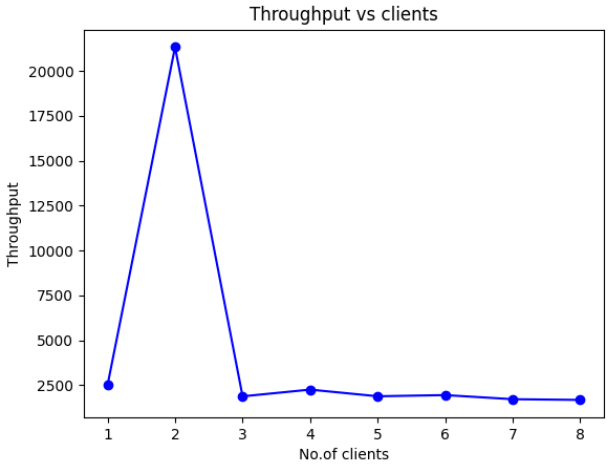


Fig. 7. Clients vs Throughput

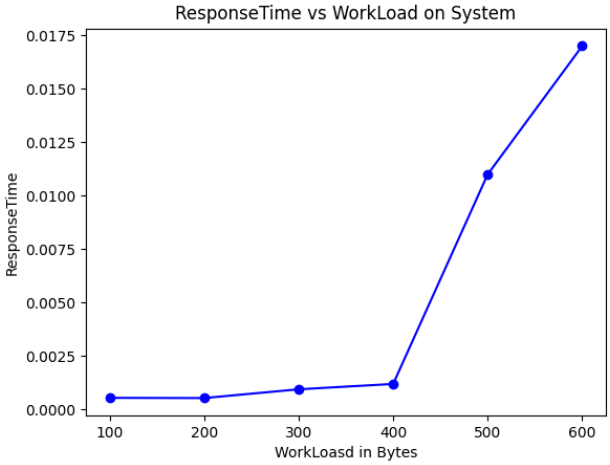


Fig. 8. Workload vs Response time

VII. Conclusion

: We are exploring the implementation of a backup server sys- tem to enhance node failure and disconnection management. This system aims to automatically redirect connections to corresponding backup nodes when a primary node is switched off. The envisioned setup includes the creation of backup servers for each node, providing a seamless transition for connecting clients and ensuring continuous system availability even in the face of node failures.

As of now we have included backup database for every fileserver to store the current status of the fileserver before turning off and retrieving back the previous state information of the server after turning on.