Distributed File System

Rishi Borkar

August 4th, 2023

Abstract

This paper presents an implementation of a Distributed File System using Python, XML-RPC, and the Bully Algorithm for leader election. The system allows multiple nodes to share and access files across a network, providing transparent file access and manipulation. The leader election algorithm ensures consistency and efficient task coordination among the nodes. The system demonstrates fault tolerance through chunk replication, allowing files to be fully retrieved even with up to 50% nodes down. The paper also discusses scalability and suggests improvements for future work, including dynamic onboarding, Byzantine fault tolerance, and file editing capabilities. Experimental results showcase the average speeds for file creation, deletion, and retrieval based on file size. Overall, this Distributed File System implementation offers valuable insights and provides a foundation for further research and development in the field of distributed computing.

1 Introduction

A distributed file system is a network-based file system that allows multiple computers to share and access files and data across a network. It provides a transparent view of the files and their organization, enabling users to access and manipulate files as if they were stored on their local machine.

The first distributed file system called the Newcastle Connection was developed in the 1970s at the University of Newcastle. It formed the forerunner to the Network File System (NFS) which was developed in the 1980s by Sun Microsystems. NFS is still widely used today as the underlying distributed system for various file systems. In addition to the features present in the Newcastle Connection, NFS also provides scalability and fault tolerance. In the 2000s, several other distributed file systems were developed, including the Google File System (GFS), the Hadoop Distributed File System (HDFS), and the Amazon Elastic File System (EFS). These file systems were designed to meet the needs of large-scale distributed applications, such as web search and cloud computing. Today, distributed file systems are an essential part of modern computing infrastructure. They are used by businesses and organizations to store and share large amounts of data.

2 Leader Election

In distributed computing, leader election is needed in order to maintain consistency in the system. The leader acts as an organizer of the task assigned to the system and is elected by the network nodes by communicating among themselves in order to decide which of them will get into the "leader" state. There are many leader election algorithms, each with its own strengths and weaknesses. Some of the most common leader election algorithms include:

- Bully algorithm: This is a simple and efficient algorithm that is well-suited for synchronous systems. The Bully algorithm works by having each node with a unique ID number. When a node starts up or the current leader fails, the node with the highest ID number becomes the leader.
- Ring algorithm: This algorithm is similar to the Bully algorithm, but it is designed for asynchronous systems. The Ring algorithm works by having each node in the system form a ring with its neighbors. When a node starts up or the current leader fails, the node at the head of the ring becomes the leader.
- Paxos algorithm: This is a more complex algorithm that is designed for asynchronous systems. The Paxos algorithm works by having each node in the system propose a leader. If a majority of nodes agree on the same leader, that node becomes the leader. If no majority can be reached, the algorithm will continue until a leader is elected. It is used by many distributed systems, including ZooKeeper and Google Spanner.
- Raft algorithm: This is a more recent algorithm that is similar to the Paxos algorithm. The Raft algorithm works by having each node in the system vote for a leader. If a majority of nodes vote for the same leader, that node becomes the leader. If no majority can be reached, the algorithm will continue until a leader is elected. The algorithm followed by Raft to achieve this differs and is simpler than Paxos. It is used in distributed systems such as etcd in Kubernetes and Amazon ECS.

Choosing the best leader election algorithm for a particular system depends on the specific requirements of the system. For example, if the system needs to be able to elect a leader quickly, then the Bully algorithm is a good choice. If the system needs to be able to elect a leader in an asynchronous system, then the Paxos algorithm or the Raft algorithm is a better choice.

Leader election is an important concept in distributed systems because it allows for the efficient coordination of tasks among multiple nodes. By electing a single leader, the system can avoid the overhead of having to coordinate tasks among all of the nodes. This can improve the performance and scalability of the system.

3 Implementation

The system has been implemented in Python and the source code is available on Github. The system uses XMLRPC to communicate using HTTP and the files are transfered as binary objects. It has create, delete and read file functionality.

Leader Election The system uses the Bully Algorithm for Leader election that has been implemented to follow the steps shown below.

1. Initially, the node with the highest ID is chosen as the leader.
2. Each node sends a heartbeat periodically to the leader after random intervals to check if the leader is still up.
3. If node P does not receive an acknowledgment of its heartbeat from the current leader, then P assumes that the leader has failed.
4. P then sends an election message to all nodes with a higher ID than its own.
5. If P receives a reply to its election message from a node with a higher ID, then P drops out of the election.
6. If P does not receive a reply to its election message from a process with a higher ID, then P becomes the new leader, broadcasts that it is the leader to all other nodes and the algorithm restarts from Step 2.

The decision to have each node sends the heartbeat to the leader rather than the leader sending a heartbeat to each node was made to allow for easier implementation and to allow for situations in which just one node finds out that the leader is down rather than all the nodes at the same time. Figure 1 and Figure 2 illustrate the operation of this algorithm.

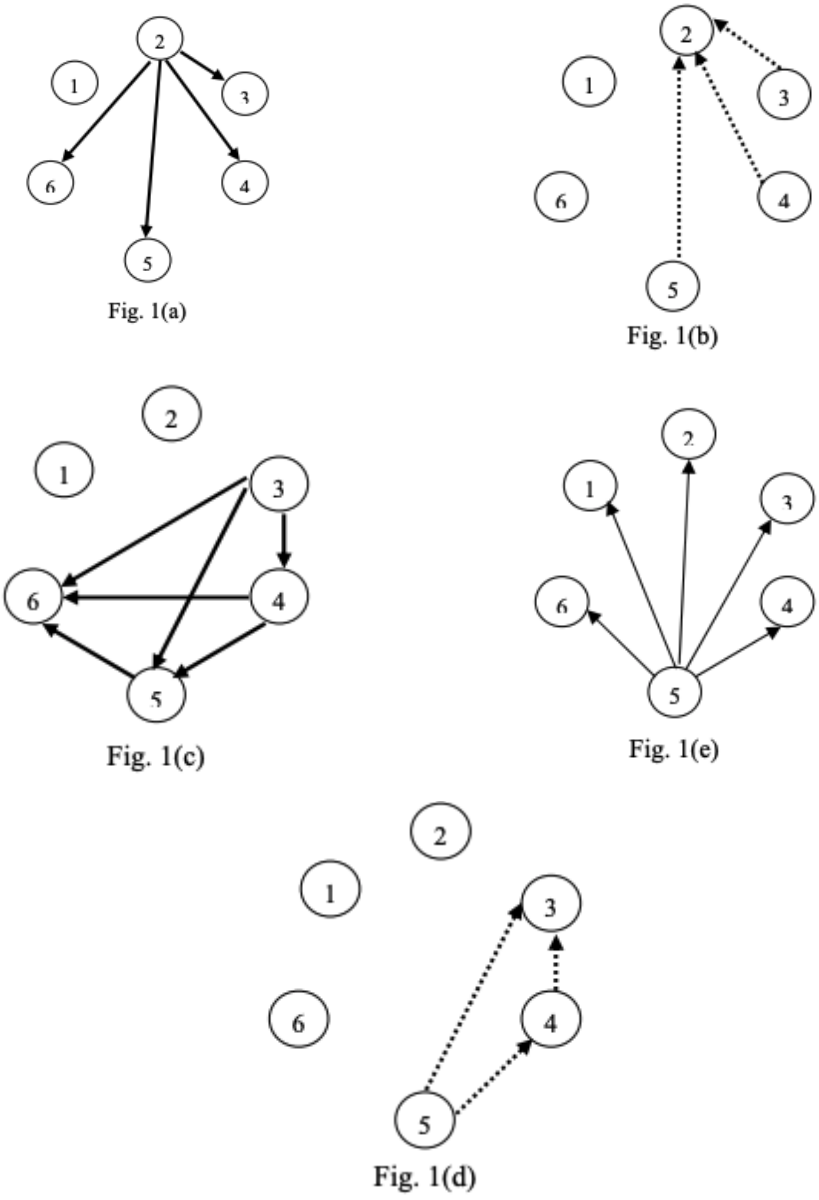

Figure 1: The bully election algorithm [1]

[1]Images from [7].

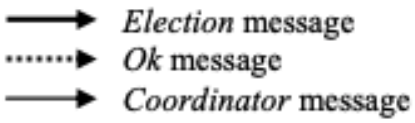

Figure 2: Types of messages [2]

File Generation For the purpose of simulation, dummy files are generated of size equal to 10,240,000 x number of characters in the filename. These files are then broken and stored into chunks of size 2.56 MB across the nodes in the system.

Transparency The files are stored and retrieved in a transparent manner and the client does not need to specify the nodes that chunks need to be retrieved from. The ledger that stores the file chunk locations is maintained with the leader and retrieved by the other nodes to maintain consistency.

Fault Tolerance An algorithm is used to break up the file into chunks and evenly distribute them among the nodes with sufficient overlap such that at least 2 copies of a chunk are maintained over two nodes in the system. When the file is being accessed, only one copy of the chunk is retrieved. If a chunk retrieval fails, one of its copies gets retrieved as a backup. A file in the system can still be completely retrieved when up to 50% of the nodes are down due to chunk replication.

Scalability For this implementation the system has only been set up to allow for the connection of 4 nodes with fixed IPs which get pulled from a local configuration file. This can be increased by setting up an initial server that every node connects to when they join the network and gets assigned an ID and IP dynamically. This will allow several nodes to be able to join and leave the network. Figure 3 shows the time taken for the system to create, delete or view/retrieve a file based on its size. The fluctuations in time taken to view a file can be attributed to how many chunks of the file need to be accessed from another node. During retrieval, local chunks are prioritized and are accessible faster.

[2]Image from [7].

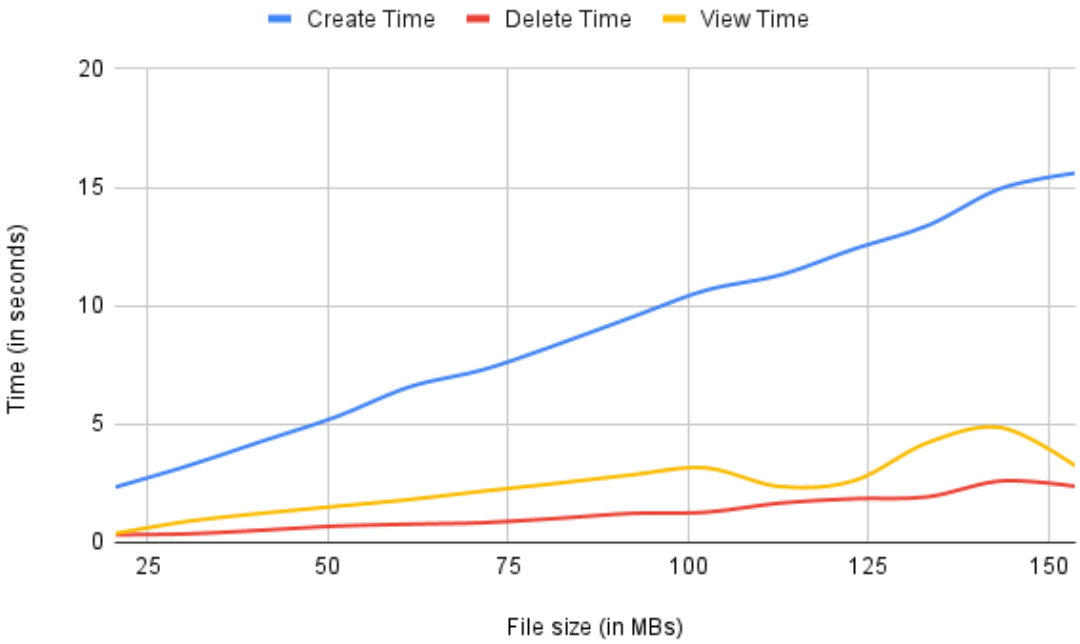

Figure 3: Create file, delete file, and view file time based on file size

Table 1 below shows the average speeds provided by the system for create, delete, and view file operations.

	Average Speed
Create File	9.64 MBps
Delete File	72.97 MBps
View File	37.32 MBps

Table 1: Average Speeds

4 Future Work

Additional work in this project would involve the implementation of an initial onboarding server that dynamically assigns nodes with their ID and forwards their IP to the current leader of the system to be added to the map. If there is no leader, the new joining node automatically becomes the leader. In this case, when the nodes in the network are randomly joining and leaving, the server

shall also have an active service that replicates chunks across the network to maintain availability. In this case, the trustworthiness of every node can be determined by its uptime in the network.

This implementation also assumes that all the nodes are trustworthy. To allow for Byzantine Fault Tolerance (when some of the nodes act maliciously), two changes need to be implemented:

- The hash of each individual file chunk can be stored with the leader. Whenever a file is retrieved, it is checked against the hash to ensure that it is unchanged. In case a difference is found, a replica of the chunk can be used instead.
- Rather than using the Bully Algorithm for Leader election, Proof of Work or Proof of Stake can be used to decide the leader that gets to make the next change to the file ledger. Since the changes in the file ledger will then be stored on a Blockchain, any retroactive malicious changes will easily be detected and rejected.

The system currently has only create, delete and read functionality. Editing of the file can also be implemented by maintaining a mutex for each file on the leader so as to prevent multiple nodes from trying to edit a file at the same time.

References

[1] *Google*. URL: `https://groups.google.com/g/erlang-programming/c/kcQKt8-mWe8?pli=1`.

[2] *Wikipedia*, Jun 2023. URL: `https://en.wikipedia.org/wiki/Clustered_file_system#Distributed_file_systems`.

[3] *Wikipedia*, Mar 2023. URL: `https://en.wikipedia.org/wiki/Newcastle_Connection`.

[4] *Wikipedia*, Jul 2023. URL: `https://en.wikipedia.org/wiki/Paxos_(computer_science)`.

[5] *Wikipedia*, Jul 2023. URL: `https://en.wikipedia.org/wiki/Raft_(algorithm)`.

[6] Diwakar Bhardwaj. Raft and paxos: A brief introduction to the basic consensus protocols powering distributed systems..., Feb 2020. URL: `https://medium.com/the-sixt-india-blog/raft-and-paxos-a-brief-introduction-to-the-basic-consensus-protocols-powering-distributed-systems-1a0ef7ca3acb`.

[7] Quazi Ehsanul Kabir Mamun, Salahuddin Mohammad Masum, and Mohammad Abdur Rahim Mustafa. Modified bully algorithm for electing coordinator in distributed systems. *WSEAS Transactions on Computers*, 3(4):948–953, 2004.