Table of Content

Section #1 - Introduction

Nowadays, people prefer storing their personal files including documents, photos on the cloud rather than on their own computers. A cloud file management system must provide basic functionalities to its user while overcoming many challenges. Common challenges of cloud file management system include fault tolerance, task dispatching, concurrency handling, corrupted data handling, and secure sharing.

The objective of this project is to build a robust cloud file management system which can provide the basic functionalities to its users such as file uploading, file downloading, file deletion, and file searching, while resolving the all these common challenges from scratch. In this project, a cluster of file sharing server which includes one leader node and three data nodes has been built. The leader node is in charge of accepting the request from outside of the cluster and dispatching the task to the individual data node. Data node is connected to and controlled by the leader node. Data node is the substantial node who stores the users' files on its local file system and database.

The main technologies used to accomplish all these goals are Protobuf and gRPC. gRPC was used as the mechanism for communication within and outside the cluster. Protobuf was used as the messaging protocol within and outside the cluster[1,2]. Leader node and three data node was built with Java. Besides, every data node has a corresponding heartbeat server built with python. The heartbeat server is in charge of providing CPU, memory, and disk utilization information to the leader node to perform task dispatching. Both leader node and data node used mongoDB to record the file metadata. Lead node also records the individual data node information in mongoDB.

Section #2 - System Architecture

The overall system architecture is illustrated by the figure below. For a typical user, a request must be sent to the super node. All other nodes besides super node are transparent to the user. Super node holds the ip addresses of all the active leader node(active cluster). The data nodes under the leader node are transparent to the super node.

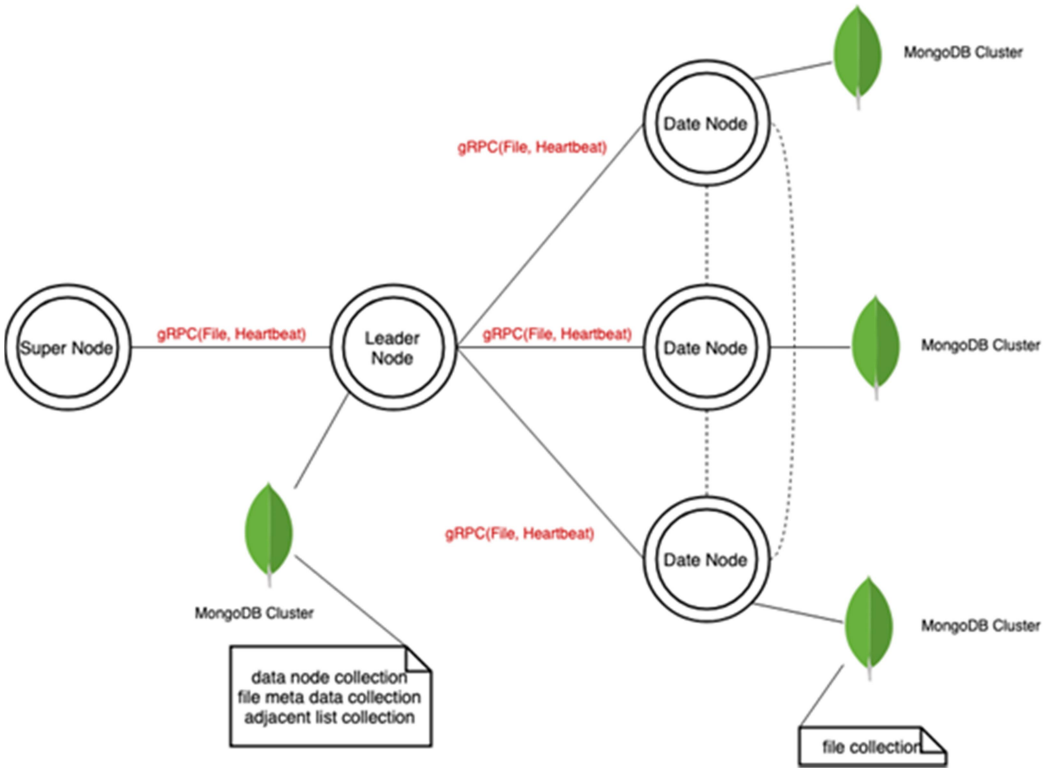


Figure. System Architecture

2.1 Super Node

The supernode works as an interface to the outside world. A user is not supposed to send a request directly to the leader nodes and data nodes. Supernode holds all the necessary information for performing task dispatching. Supernode holds the IP addresses of all leader node. In addition, the supernode has the CPU, memory, disk utilization information for all clusters. Supernode will pick the cluster with the lowest utilization rate and dispatch the task to it.

2.2 Leader Node

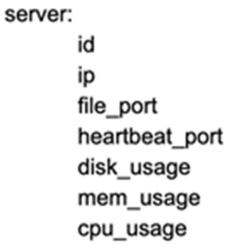
The leader node receives the request from supernode and then send the request to the data node with the lowest utilization rate to perform the task. Every leader node represents a cluster of the file server and hides the data node information from supernode. Leader node also holds all the IP address of the data node in the cluster and the utilization information of each data node.

2.3 Data Node

The data node is the substantial node who perform the users' requests. Data node receives the request from the leader node and stores or sends the user's file to the leader node. The files are stored in the data node's local file system. Data node also stores the file metadata in its local database. In addition, every data node has a heartbeat server which can help the supernode to achieve its current utilization rate.

2.4 Database Schema

The database used in this project was MongoDB. The database schemas for the leader node and data node are illustrated by the figure below. The leader node has two collections in its local database. One collection is used to record the individual data node information including the IP address, utilization rate, etc. The other collection is used to record the file metadata. More specifically, which data node is currently holding which file. Each data node has one collection in its local database. This collection is used to record the real file path of the files currently stored on it.



server_file:
username
file_name
server_id

Figure. Leader node database schema

client_file:
username
filename
path

Figure. Data node database schema

Section #3 - Functionality Design

3.1 Heartbeat & Stats Check

As mentioned in the Section#3, there are several clusters of leader node and data node in the system. It is important to frequently monitor the health of each node so that the communication between all of them can be assured to be executed. There are two mechanism for the purpose of monitoring. One is a constant Heartbeat rpc between data node and leader node i, and the other is a passive status check by super node to each registered leader node for the cluster. A illustrating diagram of the above processes is shown below.

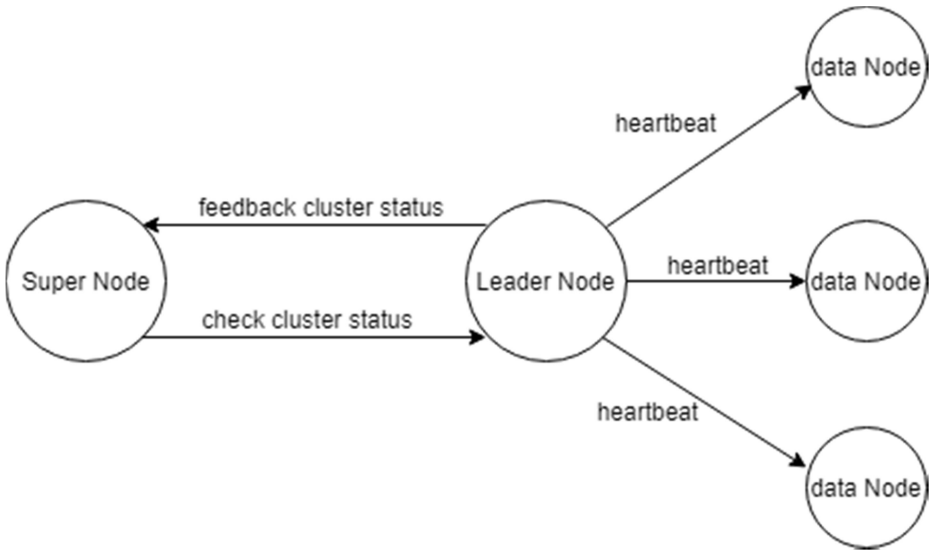


Fig.1 Heartbeat and Status Check

3.1.1 Passive status check between super node and registered leader node

When a cluster is running up and selected its leader, the leader node will send the leader node information including IP, port, and cluster name via gRPC to the super node. Super node recorded all the registered leaders for each cluster. When the super node receive a user request and needs to assign a job to a cluster, it will send request to each leader to check the current status of clusters. The status information is composed of average cpu, memory and disk usage percentage of all the data node in one cluster. If the super node does not receive a feedback on a registered cluster's status, it treats the leader node as a dead node, and will not assign the job to it.

3.1.2 Heartbeat between leader node and data node

In each cluster, leader node is responsible for managing its data node. Leader node constantly send heartbeat gRPC request to each data node in every 5 seconds, and collects data node's cpus, memory and disk usage percentage. If a data node heartbeat response is lost, leader node will treats it as a dead node, and will not assign job to it when there is.

3.2 File Upload

As a remote file management system, user can upload files from local disk to the distributed system. Super node is responsible for accepting the request, and transfer the uploaded file to a lived cluster's leader based on the status information, and recorded the file storage information of the cluster in super node. Then, the leader node will decide which data nodes to store the file

and record the file location of the data node in a cluster shared file information database. A illustrating diagram of the above processes is shown below.

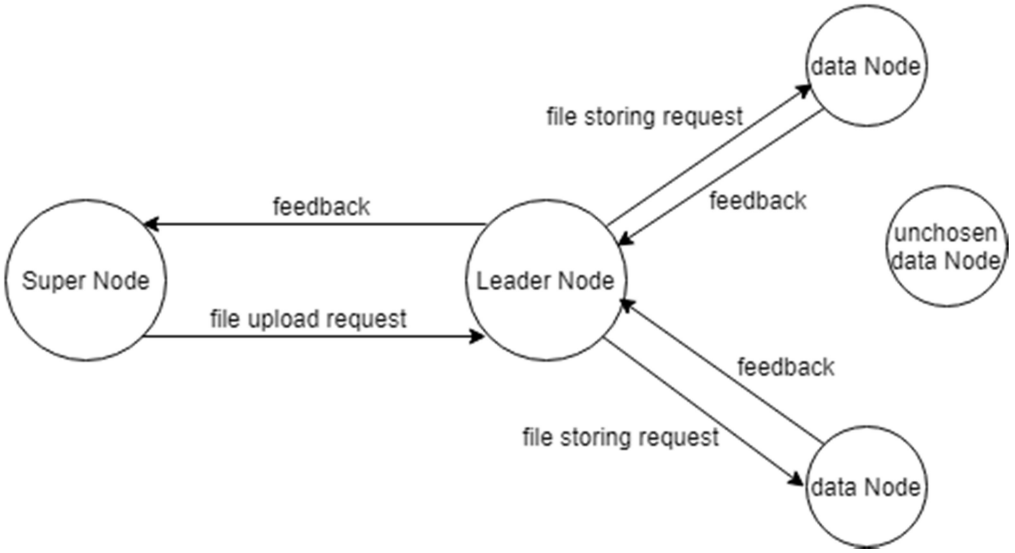


Fig.2 File Upload

3.3 File Search

A user can search a file in the remote file management system based on the file name. The super node firstly receive the request with user and file name information, then search for it in file storage record and find out the cluster that store it. A illustrating diagram of the above processes is shown below.

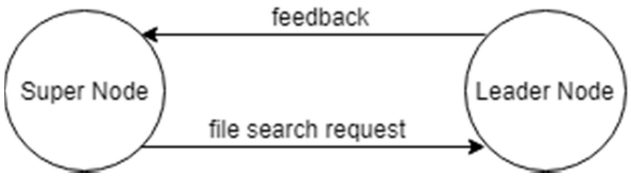


Fig.3 File Search

3.4 File List

For every user, the system can generate all the stored file information, including file name, size, data format. This is achieved by traversing file storage record in the super node and find all the files belonging to the user, and feedback all the information to end user. A illustrating diagram of the above processes is shown below.

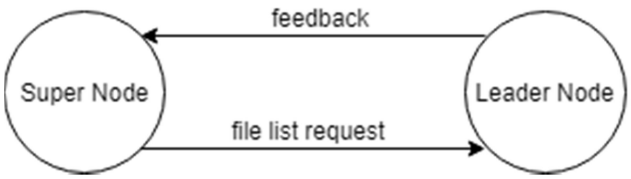


Fig.4 File List

3.5 File Download

After user get the list of all the stored files using the FileList function, he/she can select a file to download. The request is sent to super node with user and filename information. The super node firstly find out the cluster that store the file in the file storage recorde. Then it send request to the current leader node in that cluster. The leader node will find out the data node that stores the file and retrieve it to send back to super node. A illustrating diagram of the above processes is shown below.

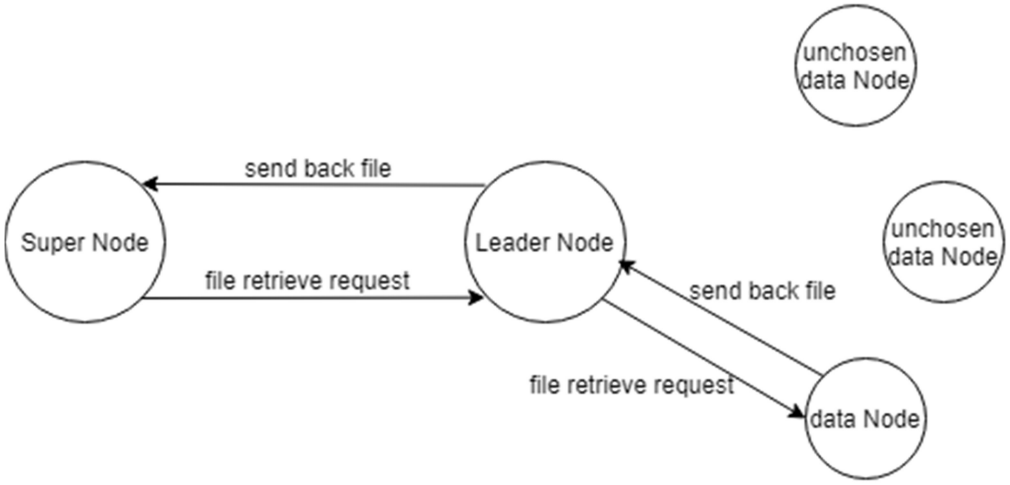


Fig.5 File Download

3.6 File Deletion

After user get the list of all the stored files using the FileList function, he/she can select a file to delete in the remote file system. The delete request is handled by super node with user and filename information. The super node figures out the cluster that store the file in the file storage record. Then it asks the current leader node in the cluster to delete the file. The leader node will request the data node that stores the file to delete the file. When the deletion is executed, data node notifies leader node the result. If it is done successfully, leader node will remove the file location record in the cluster shared database, and then notify the super node to remove the information in file storage record. A illustrating diagram of the above processes is shown below.

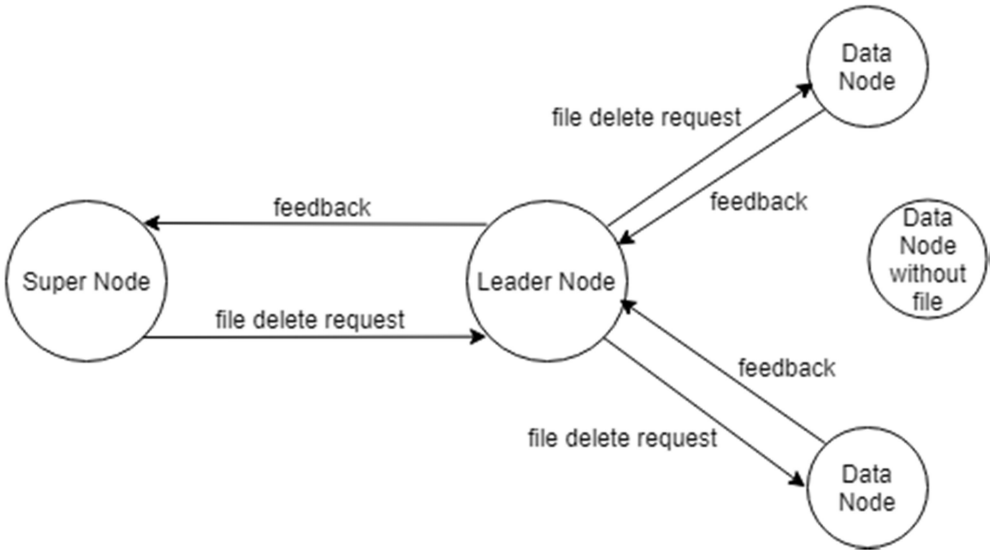


Fig.6 File Delete

3.7 Fault Tolerance(Deep Dive Component)

During this project, we have proposed two different algorithms for fault tolerance. One is called ring back up and the other is called direct replication. Direct replication was selected to be implemented as the fault tolerance algorithm for this project.

Algorithm 1 - Ring backup

In this algorithm, every data node knows its previous and next data node. In the beginning, every data node holds its original files as well as the replication files of its previous node. If one data node is down(The leader node can tell which data node is down by heartbeat packets), its next node takes its current replication files as its original files, and also take its current previous node's original file as its replication files. Its current next node will update the replication files based on the original file changes of it. The detailed process of this algorithm is illustrated by the figure below.

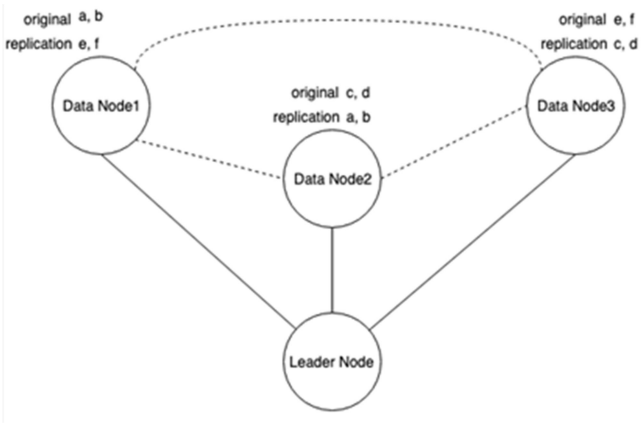


Figure. Normal mode

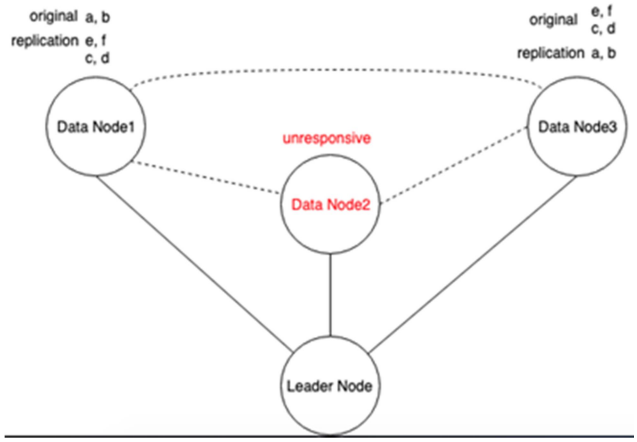


Figure. After Node2 fails

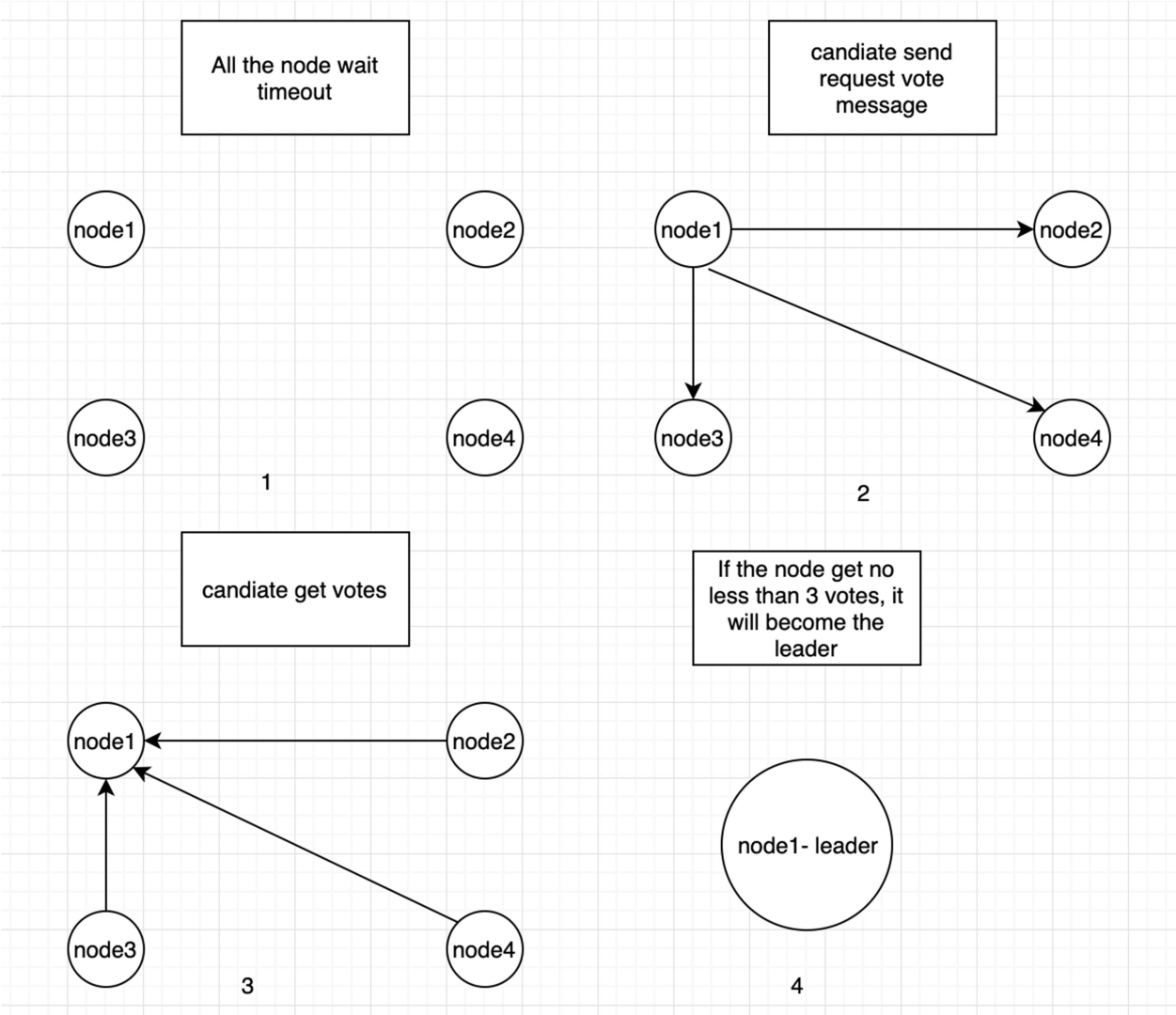
Algorithm 2: Direct replication

In this algorithm, when the leader node receives the storing request from the supernode, it will select two data nodes with the lowest utilization rate(CPU uses, Memory uses, Disk uses) to make two copies of the same file. The leader node has the information which two data nodes hold the same file. When the leader node notices that one data node is down, it will perform the requests based on the other copy. In addition, for all files on the dead node, another copy of its files will be made and stored on another alive node. In the end, the leader node will update its file metadata accordingly

3.8 Raft

Raft is a protocol for implementing distributed consensus. We use Raft protocol to choose leader.

Firstly, all our nodes start in the follower state. The election timeout is randomized to be between 150ms and 300ms.
Secondly, after the election timeout the follower becomes a candidate and starts a election term. The node votes for itself and sends out Request Vote messages to other nodes.
Finally, the receiving node hasn't voted yet in this term votes for the candidate and the node resets its election timeout. Once a candidate has a majority of votes it becomes leader. Then the leader sends its ip to super node.



Exceptional case:
If two nodes become candidates at the same time and each candidate has 2 votes. The nodes will wait for a new election and try again. Until a leader is elected.

3.9 Load Balance

We implemented two load balancing algorithms, Weighted load balance, and Round-robin load balance.

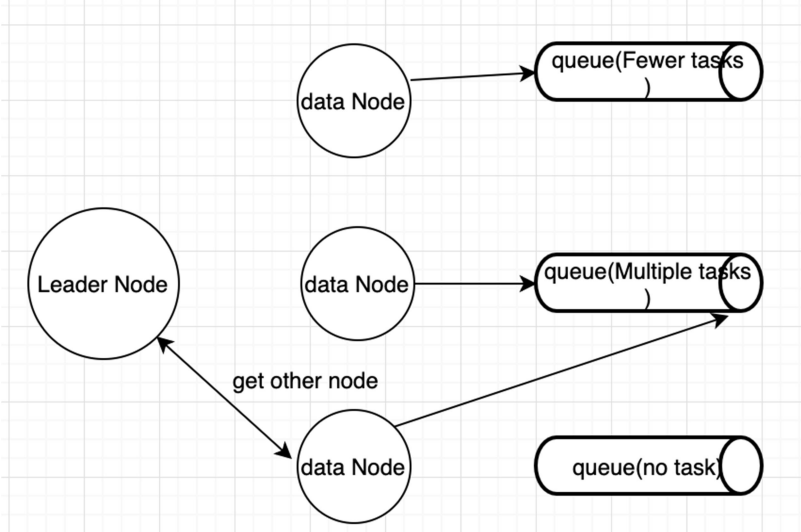
Weighted load balancing algorithm:
The leader node assigns requests according to different weights
Implementation principle: each node sends a heartbeat packet to the master node. The heartbeat packet contains the processor, memory, and hard disk information. The leader node calculates different weights based on the information, thereby allocating the request to the node with higher weight.

Round-robin load balancing algorithm:
The requests are distributed to different nodes in turn, and each service provider machine is balanced.
Implementation principle: Record the used node index by using the counter. If the last node is used, reset the counter to zero and restart the new loop.

We use the weighted load balancing algorithm. But in order to verify that our nodes work, we use round-robin to test, to avoid some node is not error, can not be accessed

3.10 Work Stealing

Each node has a double-ended queue to maintenance request. When a node completes all the requests, it sends a request to the leader node, gets the node with more tasks, then gets the task from the node and executes it.



Section #4 - Test Design & Result

4.1 System Test

i. Servers Initialization, Status Report, and Heartbeat Check

Leader Node Starts:

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_201.jdk/Contents/Home/bin/java ...
mac os x
Apr 10, 2019 11:02:57 AM com.ggsddu.fileservice.leadernode.LeaderFileServiceServer start
INFO: Server started, listening on 50051
```

Data Node 0 and 1 starts and posts node info to Leader Node:

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_201.jdk/Contents/Home/bin/java "-javaagent:/Applicat
Transfer node info success
Apr 10, 2019 11:08:07 AM com.ggsddu.fileservice.datanode.FileServiceServer start
INFO: Server started, listening on 60051
```

```
FileServiceServer
/Library/Java/JavaVirtualMachines/jdk1.8.0_181.jdk/Contents/Home/bin/java ...
Transfer node info success
Apr 10, 2019 11:06:41 AM com.ggsddu.fileservice.datanode.FileServiceServer start
INFO: Server started, listening on 60051
```

Key	Value	Type
(1) ObjectId("5cae31060b0e40651d838567")	{ 8 attributes }	Document
_id	ObjectId("5cae31060b0e40651d838567")	ObjectId
id	1	String
ip	172.20.10.8	String
file_port	60,051 (60.1K)	Int32
heartbeat_port	50,052 (50.1K)	Int32
cpu_usage	4.8	String
disk_usage	91.1	String
mem_usage	69.2	String
(2) ObjectId("5cae30b10b0e40651d838566")	{ 8 attributes }	Document
_id	ObjectId("5cae30b10b0e40651d838566")	ObjectId
id	0	String
ip	172.20.10.11	String
file_port	60,051 (60.1K)	Int32
heartbeat_port	50,052 (50.1K)	Int32
cpu_usage	3.7	String
disk_usage	81.6	String
mem_usage	56.5	String

Data Node 1 sending heartbeat to Leader Node:

```
Wills-MacBook-Pro-4:PythonHeartbeat will$ python3 heartbeat_server.py
Heartbeat requested. Disk Space: 91.4, CPU Usage: 30.4, Mem: 65.4
Heartbeat requested. Disk Space: 91.4, CPU Usage: 5.6, Mem: 65.1
Heartbeat requested. Disk Space: 91.4, CPU Usage: 15.4, Mem: 65.7
Heartbeat requested. Disk Space: 91.4, CPU Usage: 12.8, Mem: 69.0
Heartbeat requested. Disk Space: 91.4, CPU Usage: 8.0, Mem: 69.7
Heartbeat requested. Disk Space: 91.4, CPU Usage: 14.9, Mem: 69.1
Heartbeat requested. Disk Space: 91.4, CPU Usage: 5.5, Mem: 68.4
Heartbeat requested. Disk Space: 91.4, CPU Usage: 4.8, Mem: 66.7
Heartbeat requested. Disk Space: 91.4, CPU Usage: 11.0, Mem: 65.7
Heartbeat requested. Disk Space: 91.4, CPU Usage: 9.0, Mem: 65.7
Heartbeat requested. Disk Space: 91.4, CPU Usage: 4.9, Mem: 65.7
Heartbeat requested. Disk Space: 91.4, CPU Usage: 4.2, Mem: 65.2
Heartbeat requested. Disk Space: 91.4, CPU Usage: 8.1, Mem: 65.2
Heartbeat requested. Disk Space: 91.4, CPU Usage: 21.7, Mem: 68.1
Heartbeat requested. Disk Space: 91.4, CPU Usage: 12.7, Mem: 73.8
Heartbeat requested. Disk Space: 91.4, CPU Usage: 2.4, Mem: 68.7
Heartbeat requested. Disk Space: 91.4, CPU Usage: 3.3, Mem: 67.9
Heartbeat requested. Disk Space: 91.4, CPU Usage: 30.8, Mem: 64.3
Heartbeat requested. Disk Space: 91.1, CPU Usage: 42.9, Mem: 66.9
Heartbeat requested. Disk Space: 91.1, CPU Usage: 8.3, Mem: 67.8
Heartbeat requested. Disk Space: 91.1, CPU Usage: 25.4, Mem: 65.9
Heartbeat requested. Disk Space: 91.1, CPU Usage: 27.6, Mem: 72.8
Heartbeat requested. Disk Space: 91.1, CPU Usage: 17.2, Mem: 72.1
Heartbeat requested. Disk Space: 91.1, CPU Usage: 11.5, Mem: 67.7
```

Data Node 0 sending heartbeat to Leader Node:

```
feideMacBook-Pro:PythonHeartbeat kanon$ python3 heartbeat_server.py
Heartbeat requested. Disk Space: 81.6, CPU Usage: 4.9, Mem: 56.7
Heartbeat requested. Disk Space: 81.6, CPU Usage: 8.0, Mem: 56.7
Heartbeat requested. Disk Space: 81.6, CPU Usage: 1.2, Mem: 56.7
Heartbeat requested. Disk Space: 81.6, CPU Usage: 2.5, Mem: 56.7
Heartbeat requested. Disk Space: 81.6, CPU Usage: 3.6, Mem: 56.7
Heartbeat requested. Disk Space: 81.6, CPU Usage: 1.2, Mem: 56.7
Heartbeat requested. Disk Space: 81.6, CPU Usage: 3.8, Mem: 56.7
Heartbeat requested. Disk Space: 81.6, CPU Usage: 2.4, Mem: 56.7
Heartbeat requested. Disk Space: 81.6, CPU Usage: 4.9, Mem: 56.7
Heartbeat requested. Disk Space: 81.6, CPU Usage: 7.1, Mem: 56.7
Heartbeat requested. Disk Space: 81.6, CPU Usage: 1.2, Mem: 56.7
Heartbeat requested. Disk Space: 81.6, CPU Usage: 1.2, Mem: 56.7
Heartbeat requested. Disk Space: 81.6, CPU Usage: 6.0, Mem: 56.7
Heartbeat requested. Disk Space: 81.6, CPU Usage: 1.3, Mem: 56.7
Heartbeat requested. Disk Space: 81.6, CPU Usage: 6.0, Mem: 56.7
Heartbeat requested. Disk Space: 81.6, CPU Usage: 1.2, Mem: 56.7
Heartbeat requested. Disk Space: 81.6, CPU Usage: 4.9, Mem: 56.7
Heartbeat requested. Disk Space: 81.6, CPU Usage: 8.9, Mem: 56.7
Heartbeat requested. Disk Space: 81.6, CPU Usage: 10.7, Mem: 56.7
Heartbeat requested. Disk Space: 81.6, CPU Usage: 6.1, Mem: 56.7
Heartbeat requested. Disk Space: 81.6, CPU Usage: 15.3, Mem: 56.8
Heartbeat requested. Disk Space: 81.6, CPU Usage: 7.4, Mem: 56.8
Heartbeat requested. Disk Space: 81.6, CPU Usage: 14.5, Mem: 56.8
```

ii. File Upload

The client starts and begins to send a 507.5MB movie file to the Super Node:

Username: ggsddu

Filename: test100.mov

```
// Upload Test
try {
  client.uploadFile( filepath: "/Users/will/Downloads/test/test100.mov");
  logger.info( msg: "Done with upload");
} finally {
  try {
    Thread.sleep( millis: 500);
  } catch (InterruptedException e) {
    e.printStackTrace();
  }
  client.shutdown();
}
```

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_201.jdk/Contents/Home/bin/java ...
Apr 10, 2019 11:25:33 AM com.ggsddu.fileservice.leadernode.LeaderFileServiceClient uploadFile
INFO: tid: 1, Will try to getBlob
Apr 10, 2019 11:25:33 AM com.ggsddu.fileservice.leadernode.LeaderFileServiceClient main
INFO: Done with upload
Apr 10, 2019 11:25:33 AM com.ggsddu.fileservice.leadernode.LeaderFileServiceClient$5 onNext
INFO: Client response onNext
Apr 10, 2019 11:25:33 AM com.ggsddu.fileservice.leadernode.LeaderFileServiceClient$5 onNext
INFO: File successfully uploaded
Apr 10, 2019 11:25:33 AM com.ggsddu.fileservice.leadernode.LeaderFileServiceClient$5 onCompleted
INFO: Client response onCompleted

Process finished with exit code 0
```

Leader Node metadata:

(Data Node 0 and 1 are selected to save the uploaded file)

Key	Value	Type
(1) ObjectId("5cae351d0b0e40651d83856a")	{ 5 attributes }	Document
_id	ObjectId("5cae351d0b0e40651d83856a")	ObjectId
username	ggsddu	String
filename	test100.mov	String
server_id	0	String
backup_server_id	1	String

Data Node 0 metadata:

client_file 0.011 s 1 Doc 20 p. 1 1 - 1 Tree

Key	Value	Type
(1) ObjectId("5cae35289b42413b7b9c2fd5")	{ 4 attributes }	Document
_id	ObjectId("5cae35289b42413b7b9c2fd5")	ObjectId
username	ggsddu	String
filename	test100.mov	String
path	./node_files/ggsddu/test100.mov	String

Data Node 1 metadata:

Key	Value	Type
(1) ObjectId("5cae35278bbaa95d6be02bf0")	{ 4 attributes }	Document
_id	ObjectId("5cae35278bbaa95d6be02bf0")	ObjectId
username	ggsddu	String
filename	test100.mov	String
path	./node_files/ggsddu/test100.mov	String

iii. File List

List all file saved by user ggsddu:

```
// File List Test
try {
    client.fileList(USERNAME);
    logger.info( msg: "Done with filelist");
} finally {
    try {
        Thread.sleep( millis: 500);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    client.shutdown();
}
```

Returned file list:

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_201.jdk/Contents/Home/bin/java ...
Apr 10, 2019 12:46:03 PM com.ggsddu.fileservice.leadernode.LeaderFileServiceClient main
INFO: Done with filelist
test101.m4v,test102.m4v,test100.mov,test101.mov,test102.mov

Process finished with exit code 0
```

iv. File Search

Search the test100.mov file saved for ggsddu user:

```
// File Search Test
try {
    client.fileSearch(USERNAME,  fileName: "test100.mov");
    logger.info( msg: "Done with file search");
} finally {
    try {
        Thread.sleep( millis: 500);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    client.shutdown();
}
```

Returned true and “has file” message:

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_201.jdk/Contents/Home/bin/java ...
Apr 10, 2019 12:51:31 PM com.ggsddu.fileservice.leadernode.LeaderFileServiceClient main
INFO: Done with file search
has file

Process finished with exit code 0
```

v. File Download

```
// Download Test
try {
  client.downloadFile( filename: "test100.mov");
  logger.info( msg: "Done with download");
} finally {
  try {
    Thread.sleep( millis: 500);
  } catch (InterruptedException e) {
    e.printStackTrace();
  }
  client.shutdown();
}
```

Client starts to download:

Username: ggsddu

Filename: test100.mov

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_201.jdk/Contents/Home/bin/java ...
Apr 10, 2019 12:34:39 PM com.ggsddu.fileservice.leadernode.LeaderFileServiceClient main
INFO: Done with download
begin get info
begin get info
begin get info
begin get info
begin get info
```

```
begin get info
begin get info
begin get info
begin get info
begin get info
begin get info
begin get info
begin get info
begin get info
begin get info

Process finished with exit code 0
```

Since test100.mov was saved to data node 0 and data node 1, the leader node will pick the best data node as the download server for sending data. In this case, data node 0 was selected.

The output from Data Node 0:

```
FileServiceServer
onNext count: 91
onNext count: 92
onNext count: 93
onNext count: 94
onNext count: 95
onNext count: 96
onNext count: 97
onNext count: 98
onNext count: 99
onNext count: 100
onNext count: 101
onNext count: 102
onNext count: 103
download success
```

vi. File Delete

Delete test100.mov:

```
// File Delete Test
try {
    client.fileDelete(USERNAME, fileName: "test102.mov");
    logger.info( msg: "Done with delete");
} finally {
    try {
        Thread.sleep( millis: 500);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    client.shutdown();
}
```

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_201.jdk/Contents/Home/bin/java ...
Apr 10, 2019 12:59:06 PM com.ggsddu.fileservice.leadernode.LeaderFileServiceClient main
INFO: Done with delete
delete file

Process finished with exit code 0
```

Database before delete:

Key	Value	Type
(1) ObjectId("5cae46b00b0e40651d83856c")	{ 5 attributes }	Document
_id	ObjectId("5cae46b00b0e40651d83856c")	ObjectId
username	ggsddu	String
filename	test102.mov	String
server_id	0	String
backup_server_id	1	String
(2) ObjectId("5cae46540b0e40651d83856b")	{ 4 attributes }	Document
(3) ObjectId("5cae351d0b0e40651d83856a")	{ 5 attributes }	Document
(4) ObjectId("5cae34d30b0e40651d838569")	{ 5 attributes }	Document
(5) ObjectId("5cae33e90b0e40651d838568")	{ 5 attributes }	Document

Database after delete:

Key	Value	Type
(1) ObjectId("5cae46540b0e40651d83856b")	{ 4 attributes }	Document
(2) ObjectId("5cae351d0b0e40651d83856a")	{ 5 attributes }	Document
(3) ObjectId("5cae34d30b0e40651d838569")	{ 5 attributes }	Document
(4) ObjectId("5cae33e90b0e40651d838568")	{ 5 attributes }	Document

Data Node 0 output:

```
recieve delete file request.
test102.mov is deleted!
delete count 1
```

Data Node 1 output:

```
recieve delete file request.
test102.mov is deleted!
delete count 1
```

vii. Fault Tolerance

Shutdown Data Node 0, keep Data Node 1:

(heartbeat info for Data Node 0 is automatically set to null)

Key	Value	Type
(1) ObjectId("5cae31060b0e40651d838567")	{ 8 attributes }	Document
_id	ObjectId("5cae31060b0e40651d838567")	ObjectId
id	1	String
ip	172.20.10.8	String
file_port	60,051 (60.1K)	Int32
heartbeat_port	50,052 (50.1K)	Int32
cpu_usage	3.4	String
disk_usage	92.3	String
mem_usage	62.4	String
(2) ObjectId("5cae30b10b0e40651d838566")	{ 8 attributes }	Document
_id	ObjectId("5cae30b10b0e40651d838566")	ObjectId
id	0	String
ip	172.20.10.11	String
file_port	60,051 (60.1K)	Int32
heartbeat_port	50,052 (50.1K)	Int32
cpu_usage	null	Null
disk_usage	null	Null
mem_usage	null	Null

Try filelist for user ggsddu:

```
// File List Test
try {
    client.fileList(USERNAME);
    logger.info( msg: "Done with filelist");
} finally {
    try {
        Thread.sleep( millis: 500);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    client.shutdown();
}
```

The system is still available for listing files:

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_201.jdk/Contents/Home/bin/java ...
Apr 10, 2019 1:04:19 PM com.ggsddu.fileservice.leadernode.LeaderFileServiceClient main
INFO: Done with filelist
test101.m4v,test102.m4v,test100.mov,test101.mov

Process finished with exit code 0
```

Try download file test100.mov:

```java
// Download Test
try {
  client.downloadFile( filename: "test100.mov");
  logger.info( msg: "Done with download");
} finally {
  try {
    Thread.sleep( millis: 500);
  } catch (InterruptedException e) {
    e.printStackTrace();
  }
  client.shutdown();
}
```

The file is still available for download:

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_201.jdk/Contents/Home/bin/java ...
Apr 10, 2019 1:07:58 PM com.ggsddu.fileservice.leadernode.LeaderFileServiceClient main
INFO: Done with download
begin get info
begin get info
begin get info
begin get info
```

```
begin get info
begin get info

Process finished with exit code 0
```

viii. Network failure and Recovery

When disconnecting the LAN cable of a Data Node, Leader Node will get an exception which says “Host is down: /172.20.10.11:50052”.

```
Apr 10, 2019 1:11:04 PM com.ggsddu.fileservice.leadernode.HeartbeatClient getClusterStats
WARNING: RPC failed: Status{code=UNAVAILABLE, description=io exception, cause=io.grpc.netty.shaded.io.netty.channel.AbstractChannel$AnnotatedSocketException: Host is down: /172.20.10.11:50052
    at sun.nio.ch.Net.connect0(Native Method)
    at sun.nio.ch.Net.connect(Net.java:454)
    at sun.nio.ch.Net.connect(Net.java:446)
    at sun.nio.ch.SocketChannelImpl.connect(SocketChannelImpl.java:648)
    at io.grpc.netty.shaded.io.netty.util.internal.SocketUtils$3.run(SocketUtils.java:83)
    at io.grpc.netty.shaded.io.netty.util.internal.SocketUtils$3.run(SocketUtils.java:80) <1 internal call>
    at io.grpc.netty.shaded.io.netty.util.internal.SocketUtils.connect(SocketUtils.java:80)
    at io.grpc.netty.shaded.io.netty.channel.socket.nio.NioSocketChannel.doConnect(NioSocketChannel.java:312)
    at io.grpc.netty.shaded.io.netty.channel.nio.AbstractNioChannel$AbstractNioUnsafe.connect(AbstractNioChannel.java:254)
    at io.grpc.netty.shaded.io.netty.channel.DefaultChannelPipeline$HeadContext.connect(DefaultChannelPipeline.java:1366)
    at io.grpc.netty.shaded.io.netty.channel.AbstractChannelHandlerContext.invokeConnect(AbstractChannelHandlerContext.java:545)
    at io.grpc.netty.shaded.io.netty.channel.AbstractChannelHandlerContext.connect(AbstractChannelHandlerContext.java:530)
    at io.grpc.netty.shaded.io.netty.channel.ChannelDuplexHandler.connect(ChannelDuplexHandler.java:50)
    at io.grpc.netty.shaded.io.grpc.netty.WriteBufferingAndExceptionHandler.connect(WriteBufferingAndExceptionHandler.java:136)
    at io.grpc.netty.shaded.io.netty.channel.AbstractChannelHandlerContext.invokeConnect(AbstractChannelHandlerContext.java:545)
    at io.grpc.netty.shaded.io.netty.channel.AbstractChannelHandlerContext.access$1000(AbstractChannelHandlerContext.java:38)
    at io.grpc.netty.shaded.io.netty.channel.AbstractChannelHandlerContext$11.run(AbstractChannelHandlerContext.java:535)
    at io.grpc.netty.shaded.io.netty.util.concurrent.AbstractEventExecutor.safeExecute(AbstractEventExecutor.java:163)
    at io.grpc.netty.shaded.io.netty.util.concurrent.SingleThreadEventExecutor.runAllTasks(SingleThreadEventExecutor.java:404)
    at io.grpc.netty.shaded.io.netty.channel.nio.NioEventLoop.run(NioEventLoop.java:474)
    at io.grpc.netty.shaded.io.netty.util.concurrent.SingleThreadEventExecutor$5.run(SingleThreadEventExecutor.java:909)
    at io.grpc.netty.shaded.io.netty.util.concurrent.FastThreadLocalRunnable.run(FastThreadLocalRunnable.java:30)
    at java.lang.Thread.run(Thread.java:748)
Caused by: java.net.SocketException: Host is down
    ... 24 more
}
```

After the Data Node is online again, Leader Node will be able to get its heartbeat info:

```
CPU:6.3 MEM:61.6 DISK:92.0
```

4.2 Cross-Team Integration Test

i. Servers Initialization, Status Report, and Heartbeat Check

Super Node start:

```
Wills-MacBook-Pro-4:SuperNode will$ python3 superNode.py
Supernode started on 192.168.0.9:9000
```

Leader Node starts and constantly sends cluster leader info to Super Node:

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_201.jdk/Contents/Home/bin/java ...
mac os x
Apr 10, 2019 10:10:57 AM com.ggsddu.fileservice.leadernode.LeaderFileServiceServer start
INFO: Server started, listening on 50051
Thread for super node communicating running
Post super node feedback : true
Super node complete!
Post super node feedback : true
Super node complete!
```

Super Node receives leader node info:

```
Wills-MacBook-Pro-4:SuperNode will$ python3 superNode.py
Supernode started on 192.168.0.9:9000
getLeaderInfo Called
ClusterLeaders:  {'GGsDDu-cluster': '192.168.0.10:50051'}
getLeaderInfo Called
ClusterLeaders:  {'GGsDDu-cluster': '192.168.0.10:50051'}
getLeaderInfo Called
ClusterLeaders:  {'GGsDDu-cluster': '192.168.0.10:50051'}
```

Data Node starts and posts node info to Leader Node:

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_201.jdk/Contents/Home/bin/java ...
Transfer node info success
Apr 10, 2019 10:18:44 AM com.ggsddu.fileservice.datanode.FileServiceServer start
INFO: Server started, listening on 60051
```

Data Node heartbeat service starts and constantly send heartbeat info to Leader Node:

```
Wills-MacBook-Pro-4:PythonHeartbeat will$ python3 heartbeat_server.py
Heartbeat requested. Disk Space: 90.7, CPU Usage: 22.5, Mem: 64.8
Heartbeat requested. Disk Space: 90.7, CPU Usage: 3.1, Mem: 64.5
Heartbeat requested. Disk Space: 90.7, CPU Usage: 2.4, Mem: 64.5
Heartbeat requested. Disk Space: 90.7, CPU Usage: 3.2, Mem: 64.5
Heartbeat requested. Disk Space: 90.7, CPU Usage: 0.0, Mem: 64.5
Heartbeat requested. Disk Space: 90.7, CPU Usage: 4.7, Mem: 64.7
Heartbeat requested. Disk Space: 90.7, CPU Usage: 0.8, Mem: 64.8
Heartbeat requested. Disk Space: 90.7, CPU Usage: 7.9, Mem: 64.7
Heartbeat requested. Disk Space: 90.7, CPU Usage: 1.7, Mem: 64.8
Heartbeat requested. Disk Space: 90.7, CPU Usage: 2.4, Mem: 64.5
Heartbeat requested. Disk Space: 90.7, CPU Usage: 0.8, Mem: 64.5
Heartbeat requested. Disk Space: 90.7, CPU Usage: 0.8, Mem: 64.6
```

Leader Node Database for the list of Data Node:

Key	Value	Type
(1) ObjectId("5cae25fec0428a776ccfc27d")	{ 8 attributes }	Document
_id	ObjectId("5cae25fec0428a776ccfc27d")	ObjectId
id	0	String
ip	172.20.10.8	String
file_port	60,051 (60.1K)	Int32
heartbeat_port	50,052 (50.1K)	Int32
cpu_usage	15.3	String
disk_usage	90.8	String
mem_usage	70.3	String

ii. File Upload

The client starts and begins to send a 507.5MB movie file to the Super Node:

```
// Upload Test
try {
  client.uploadFile( filepath: "/Users/will/Downloads/test/test101.m4v");
  logger.info( msg: "Done with upload");
} finally {
  try {
    Thread.sleep( millis: 500);
  } catch (InterruptedException e) {
    e.printStackTrace();
  }
  client.shutdown();
}
```

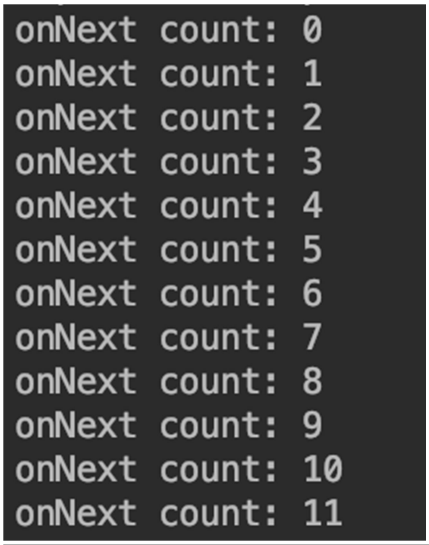
test101.m4v Apr 8, 2019 at 1:20 PM 507.5 MB Movie file

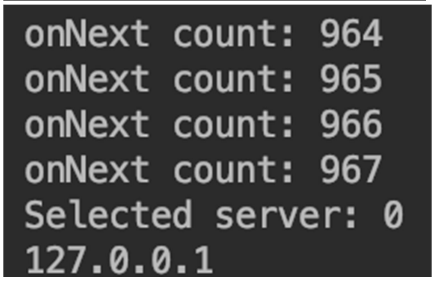
```
/Library/Java/JavaVirtualMachines/jdk1.8.0_201.jdk/Contents/Home/bin/java ...
Apr 10, 2019 10:33:14 AM com.ggsddu.fileservice.leadernode.LeaderFileServiceClient uploadFile
INFO: tid: 1, Will try to getBlob
Apr 10, 2019 10:33:15 AM com.ggsddu.fileservice.leadernode.LeaderFileServiceClient main
INFO: Done with upload
Apr 10, 2019 10:33:18 AM com.ggsddu.fileservice.leadernode.LeaderFileServiceClient$5 onNext
INFO: Client response onNext
Apr 10, 2019 10:33:18 AM com.ggsddu.fileservice.leadernode.LeaderFileServiceClient$5 onNext
INFO: File successfully uploaded

Process finished with exit code 0
```

Super Node received the file and forwards to Leader Node:

```
getLeaderInfo Called
ClusterLeaders:  {'GGsDDu-cluster': '127.0.0.1:50051'}
getLeaderInfo Called
ClusterLeaders:  {'GGsDDu-cluster': '127.0.0.1:50051'}
getLeaderInfo Called
ClusterLeaders:  {'GGsDDu-cluster': '127.0.0.1:50051'}
getLeaderInfo Called
ClusterLeaders:  {'GGsDDu-cluster': '127.0.0.1:50051'}
Inside Server method ---------- UploadFile
Node found is: 127.0.0.1:50051
```

Leader Node receives data and forwards to selected Data Nodes:





Leader Node metadata for files:

Key	Value	Type
(1) ObjectId("5cae2940c0428a776ccfc280")	{ 4 attributes }	Document
_id	ObjectId("5cae2940c0428a776ccfc280")	ObjectId
username	ggsddu	String
filename	test101.m4v	String
server_id	0	String

Data Node receives data:

```
onNext count: 954
onNext count: 955
onNext count: 956
onNext count: 957
onNext count: 958
onNext count: 959
onNext count: 960
onNext count: 961
onNext count: 962
onNext count: 963
onNext count: 964
onNext count: 965
onNext count: 966
onNext count: 967
```

Data Node metadata for files:

Key	Value	Type
(1) ObjectId("5cae29439a9f300aafb65940")	{ 4 attributes }	Document
_id	ObjectId("5cae29439a9f300aafb65940")	ObjectId
username	ggsddu	String
filename	test101.m4v	String
path	./node_files/ggsddu/test101.m4v	String

iii. File Download

Begin to download:

```
// Download Test
try {
  client.downloadFile( filename: "test101.m4v");
  logger.info( msg: "Done with download");
} finally {
  try {
    Thread.sleep( millis: 500);
  } catch (InterruptedException e) {
    e.printStackTrace();
  }
  client.shutdown();
}
```

Download success:

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_201.jdk/Contents/Home/bin/java ...
Apr 10, 2019 10:54:10 AM com.ggsddu.fileservice.leadernode.LeaderFileServiceClient main
INFO: Done with download
begin get info
begin get info
begin get info
begin get info
begin get info
begin get info
begin get info
begin get info
```

```
begin get info
begin get info
begin get info

Process finished with exit code 0
```

Section #5 - System Evaluation & Improvement

5.1 Sharding

The system can be further optimized by applying sharding algorithms. When a large file is uploaded, the leader node can separate it into many small pieces and save them into multiple data nodes. This will help to reduce the workload of a single data node and distribute tasks to several sites. When a user wants to retrieve the file, the leader node will find those file pieces and return a single complete file to the user.

5.2 Traffic Improvement

Leader node is currently the single contacting point for clients. All data upload and download must go through the leader node to reach the data node. A better approach would be that the leader node is the one for saving the metadata of file locations. Client contact leader node to get the IP and port number of the corresponding data node, and it will directly download and upload files to the data node.

5.3 Secure Connection, certificate key

Currently, the system is not secure. The client can access all data from the interface. In the future, we can add a login function to bind user information. Users can only access files belonging to them, and they cannot manipulate files belonging to others.

Section #6 - Conclusion

Project Fluffy is a distributed and scalable remote file management system with high performance and availability. We succeeded in implementing the functions for the use case of file uploading, downloading, updating, search, listing, and deletion. With hardware failure and software malfunction taken into consideration, we also built the system with the capability of fault tolerance and auto recovery. By the use of open source gRPC via protocol buffer, we were able to build a more efficient communication system than the most common JSON/XML based RESTFUL HTTP technology. The key features of gRPC is public APIs defined in protocol buffer with both synchronous and Asynchronous function calls, and more compact data wiring technique by message field name replacement [3,4].

Through this project, we learned about a new and efficient RPC technology and developed a comprehensive mindset about a highly scalable and distributed system. In addition, through the system testing, we were aware of the complexity of debugging in a distributed system. This project awarded us a thorough learning experience in developing high-performance enterprise application.