Universidad Autónoma de la Ciudad de México

Proyecto Final Transmisión de Audio

Elección con Raft/Python

Sistemas Distribuidos

Profesor: Elizabeth López Lozada

MARTINEZ MÉNDEZ JONATAN IVAN

MIRANDA MORALES EMMA ADELIZ

INDICE

1. Introducción…….. 3

2. Elección con Raft…… 3

2.1 Código Servidor…..……..…… 3

2.2 Código Cliente……..…….. 7

2.3 Ejecución de los códigos...……..……. 7

2.4 Resultados ….……...…….. 11

3. Conclusiones…….. 12

4. Referencias…… 12

1. Introducción

Raft es un algoritmo de consenso diseñado como una alternativa a la familia de algoritmos Paxos. Los Paxos son una familia de protocolos que precisamente permite resolver el consenso en una red de procesadores poco fiables o falibles. El consenso implica que varios servidores se pongan de acuerdo sobre los valores. Una vez que alcanzan una decisión sobre un valor, esa decisión es definitiva. Por ejemplo, un clúster de cinco servidores puede seguir funcionando incluso si fallan dos. Si fallan más, dejan de progresar, sin embargo nunca devolverán un resultado incorrecto.

Se ha demostrado que el algoritmo Raft no solo es más comprensible que Paxos, sino no que es más seguro y ofrece algunas características adicionales. Entre ellas la forma genérica de distribuir una máquina de estado a través de un clúster de sistemas informáticos, asegurando que cada nodo del clúster esté de acuerdo con la misma serie de transiciones de estado.

Raft implementa el consenso mediante un enfoque de líder. El clúster tiene un solo líder elegido que es totalmente responsable de administrar la replicación de registros en los demás servidores del clúster. Significa que el líder puede decidir sobre la ubicación de las nuevas entradas y el establecimiento del flujo de datos entre él y los otros servidores. Un líder lidera hasta que falla o se desconecta, en cuyo caso se elige un nuevo líder.

Ahora bien, para entenderlo mejor, vamos a llevar a cabo una transmisión de Audio que se distribuya entre diferentes dispositivos. Lo que se busca, es, que a pesar de que el audio se detenga o falle en alguno de los clúster, pueda continuar con la transmisión en el resto de dispositivos.

2. Elección con Raft

Esta práctica la llevaremos a cabo en el lenguaje de programación Python, la idea consiste en crear un código que tendrá las funciones del servidor y otro que tendrá las funciones del cliente. Los cuales tendrán comunicación mediante sockets.

2.1 Código Servidor

Primero importaremos los módulos necesarios, en el caso del módulo de sockets lo vamos a utilizar para la comunicación de red, el módulo threading nos va a permitir trabajar con hilos y el módulo os nos va a proporcionar las funciones para interactuar con el sistema operativo.

```
1   import socket
2   import threading
3   import os
```

Posteriormente vamos a definir una función que se llama handle_client que toma tres argumentos: connection, address, connections. En el caso de connection va a representar

el socket de conexión entre el cliente y el servidor, address va a contener la dirección IP y el número de puerto del cliente que se ha conectado al servidor y connections es una lista que almacena las conexiones activas en el servidor. En el caso del print nos va imprimir un mensaje en la consola indicando que se ha establecido una conexión desde la dirección especificada por address.

```
def handle_client(connection, address, connections):
    print(f"Conexión establecida desde {address}")
```

En este caso creamos una variable que se llama welcome_message, el cual mostrará el mensaje “Conexión establecida. Envíe su archivo.” Posteriormente enviaremos el mensaje de bienvenida al cliente a través del socket de conexión (connection) e imprimiremos el mensaje de bienvenida al cliente.

```
try:
    # Enviar mensaje de bienvenida al cliente
    welcome_message = "Conexión establecida. Envíe su archivo."
    connection.sendall(welcome_message.encode('utf-8'))
    print("Mensaje de bienvenida enviado al cliente.")
```

Aquí estamos definiendo un marcador especial llamado EOF (End of File) para indicar el final del archivo. Cuando este marcador se encuentra en los datos recibidos, sabremos que hemos recibido todo el archivo. Después con filename = f'received_file_from_{address[0]}.bin': Estamos creando una variable llamada filename que contiene el nombre del archivo que se guardará en el servidor. El nombre del archivo incluye la dirección IP del cliente (address[0]) y la extensión .bin. Posteriormente se ejecuta el bucle while True hasta que se rompa explícitamente. Dentro del bucle recibimos los datos del cliente con if end_marker in data verificamos si el marcador EOF está presente en los datos recibidos. Si es así, eliminamos el marcador y escribimos los datos restantes en el archivo. Con if data si aún hay datos después de eliminar el marcador, los escribimos en el archivo, con break salimos del bucle y finalmente imprimimos un mensaje en la consola indicando que el archivo ha sido recibido correctamente desde la dirección IP del cliente.

```
14          # Recibir datos del archivo en bloques de 1024 bytes
15          end_marker = b'EOF'  # Marcador especial para indicar el fin del archivo
16          filename = f'received_file_from_{address[0]}.bin'
17          with open(filename, 'wb') as f:
18              while True:
19                  data = connection.recv(1024)
20                  if end_marker in data:
21                      data = data.replace(end_marker, b'')
22                      if data:
23                          f.write(data)
24                      break
25                  f.write(data)
26          print(f"Archivo recibido de {address}")
```

Este bloque de código es de los más importantes, ya que este nos permitirá cargar el

archivo en formato mp3. Con if filename.lower().endswith(('.mp3', '.wav', '.ogg')) verifica si el nombre del archivo tiene alguna de esas extensiones. Después con if os.path.exists(filename) vamos a verificar si el archivo realmente existe en el sistema de archivos. Si ambas condiciones se cumplen, se imprime "Reproduciendo archivo de música" en la consola. Luego, se intenta abrir el archivo utilizando el método os.startfile(filename). Esto abrirá el archivo con el programa predeterminado del sistema operativo. Finalmente si ocurre algún error durante la apertura del archivo, se captura la excepción y se imprime un mensaje de error.

```
28        # Reproducir el archivo si es música y si existe
29        if filename.lower().endswith(('.mp3', '.wav', '.ogg')) and os.path.exists(filename):
30            print("Reproduciendo archivo de música...")
31            try:
32                os.startfile(filename)  # Abre el archivo con el programa predeterminado de Windows
33            except Exception as e:
34                print(f"No se pudo abrir el archivo: {e}")
```

Posteriormente si el archivo se cargó correctamente enviaremos una confirmación al cliente. Aquí estamos creando una variable llamada confirmation_message y le estamos asignando el valor de la cadena de texto "Archivo recibido con éxito." Con connection enviaremos un mensaje de confirmación al cliente a través del socket de conexión y se imprime. Después creamos una excepción en caso de que ocurra algún problema e imprimiremos el mensaje en la consola. Si se produce un error, imprimimos un mensaje indicando que la conexión se ha cerrado desde la dirección IP del cliente. Finalmente con connection.close() cerramos la conexión con el cliente del connections[address], eliminamos la entrada correspondiente a la dirección IP del cliente de la lista de conexiones activas.

```
36            # Enviar confirmación al cliente
37            confirmation_message = "Archivo recibido con éxito."
38            connection.sendall(confirmation_message.encode('utf-8'))
39            print("Confirmación enviada al cliente.")
40
41        except Exception as e:
42            print(f"Error al recibir datos de {address}: {e}")
43
44        print(f"Conexión cerrada desde {address}")
45        connection.close()
46        del connections[address]
```

Aquí definimos la función llamada server() que crea el servidor de sockets. Posteriormente vamos a crear el objeto llamado server_socket. El try y el except se van a encargar de manejar los errores. Con server_socket.bind vamos a enlazar el socket del servidor a la dirección IP que nosotros vamos a ingresar, seguida del puerto. Después indicamos que el servidor está configurado para escuchar hasta 3 conexiones entrantes en cola. Esto significa que puede manejar hasta 3 clientes simultáneamente. Posteriormente se muestra un mensaje en la consola donde indicamos que el servidor se ha iniciado correctamente en el IP y puerto correspondientes. Finalmente creamos un diccionario llamado connections para almacenar las conexiones activas. Cada conexión se identificará por su dirección IP y puerto.

```python
def server():
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        server_socket.bind(('192.168.1.83', 8080))
        server_socket.listen(3)
        print("Servidor iniciado en 192.168.1.83:8080")
    except Exception as e:
        print(f"Error al intentar bind en el puerto: {e}")
        return


    connections = {}
```

Finalmente crearemos un bucle infinito. En el cual el servidor continuará esperando nuevas conexiones y manejando clientes mientras este bucle se ejecute. Con connection, address = server_socket.accept() esperaremos a que llegue una nueva conexión. Cuando un cliente se conecta al servidor, esta línea acepta la conexión entrante, devolviendo los valores del socket específico para el cliente y la dirección IP y número de puerto. Después se agrega la nueva conexión al diccionario y creamos un nuevo hilo. Finalmente con la última línea de código verificamos si el archivo se está ejecutando directamente. Si es así, se llama a la función server() para iniciar el servidor.

```python
    while True:
        print("Esperando nuevas conexiones...")
        connection, address = server_socket.accept()
        print(f"Nueva conexión aceptada de {address}")
        connections[address] = connection
        threading.Thread(target=handle_client, args=(connection, address, connections)).start()

if __name__ == "__main__":
    server()
```

2.1 Código Cliente

Primero importaremos los módulos necesarios, en el caso del módulo de sockets, nos va a permitir administrar conexiones de res. El módulo os nos va a proporcionar las funciones para interactuar con el sistema operativo y el módulo subprocess nos va a permitir invocar procesos desde Python y comunicarse con ellos.

```python
import socket
import os
import subprocess
```

Con server_IP vamos a asignaR la dirección IP, es muy importante que esta coincida con la dirección IP del servidor, de lo contrario se presentarán errores de conexión.

Posteriormente vamos a ingresar el número de puerto, que del mismo modo debe coincidir con el puerto del servidor. Finalmente estamos definiendo un marcador especial llamado 'EOF' (End of File) para indicar el final del archivo.

```
5   def main():
6       server_ip = '192.168.43.129'
7       server_port = 8080
8       end_marker = b'EOF'  # Marcador especial para indicar el fin del archivo
```

En este caso vamos a establecer una conexión con el servidor. Primero vamos a crear un objeto de socket llamado client_socket, el try y el except se encargaran de manejar alguna excepción. Con client_socket.connect((server_ip, server_port)) vamos a intentar establecer una conexión con el servidor utilizando la dirección IP server_ip y el puerto server_port. Si la conexión se establece correctamente, se imprime “Conexión establecida con el servidor.” en la consola. El return, solo actuara en caso de que se produzca algún error.

```
10      client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
11      try:
12          client_socket.connect((server_ip, server_port))
13          print("Conexión establecida con el servidor.")
14      except Exception as e:
15          print(f"No se pudo conectar al servidor: {e}")
16          return
```

En este caso vamos a crear un bucle infinito el cual permitirá recibir los mensajes del servidor, en el cual pediremos al usuario que ingrese la ruta del archivo que desea enviar. Con client_socket vamos a recibir los datos del servidor a través del socket del cliente. Después con filename = input("Ingrese la ruta del archivo a enviar (o 'exit' para salir): ") vamos a solicitar al usuario que ingrese la ruta del archivo que desea enviar al servidor o ingresar exit para salir. Finalmente if filename.lower() == 'exit': verifica si el usuario ingresó “exit”. Si es así, se rompe el bucle y salimos del programa.

```
18      while True:
19          try:
20              message = client_socket.recv(1024).decode('utf-8')
21              print("Mensaje recibido:", message)
22
23              # Subir un archivo
24              filename = input("Ingrese la ruta del archivo a enviar (o 'exit' para salir): ")
25              if filename.lower() == 'exit':
26                  break
```

Con with open abriremos el archivo especificado. Posteriormente con file_data = f.read(): leemos todo el contenido del archivo y lo almacenamos en la variable file_data. Esto carga el archivo completo en memoria. Posteriormente client_socket nos permitirá enviar todos los datos del archivo al servidor a través del socket del cliente. El end_maker indicara que hemos terminado de enviar el archivo. Finalmente, imprimimos un mensaje en la consola para indicar que el archivo se ha enviado correctamente al servidor.

```
            with open(filename, 'rb') as f:
                file_data = f.read()
                client_socket.sendall(file_data)
                client_socket.sendall(end_marker)  # Enviar el marcador especial
                print("Archivo enviado con éxito.")
```

Este bloque de codigo es de los más importantes ya que es el que se encarga de reproducir un archivo de música si cumple con ciertas condiciones. Con if filename.lower().endswith(('.mp3', '.wav', '.ogg')): se verifica si el nombre del archivo tiene una de las siguientes extensiones: .mp3, .wav o .ogg. Si es así, se considera como un archivo de música. Posteriormente si el archivo es de música, se imprime un mensaje en la consola indicando que se está reproduciendo un archivo de música. Finalmente utilizando el módulo subprocess abriremos el archivo en el reproductor predeterminado del sistema.

```
            # Reproducir el archivo si es música
            if filename.lower().endswith(('.mp3', '.wav', '.ogg')):
                print("Reproduciendo archivo de música...")
                subprocess.Popen(['xdg-open', filename])  # Abre el archivo en el reproductor predeterminado
```

Con except ConnectionError as e: se va a capturar una excepción específica, en este si se produce un problema en la conexión. Si eso pasa se imprime un mensaje en la consola de Error de conexión. Después encontramos una segunda excepción que se ejecutará con cualquier otro error no especificado. Finalmente, se cierra el socket del cliente para liberar los recursos y cerrar la conexión con el servidor. Y con if __name__ == "__main__": se verifica si el archivo se está ejecutando directamente, si es así, se llama a la función main() para iniciar el cliente.

```
            except ConnectionError as e:
                print(f"Error de conexión: {e}.")
                break
            except Exception as e:
                print(f"Error inesperado: {e}")
                break

        client_socket.close()

if __name__ == "__main__":
    main()
```

2.3 Ejecución de los códigos

Antes de iniciar con la ejecución es importante que tengamos instalada la biblioteca pygame en cada una de las maquinas con la que haremos las pruebas. Esto para poder llevar a cabo la reproducción del audio.

```
Símbolo del sistema
Microsoft Windows [Versión 10.0.22631.3593]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\adeem>pip --version
pip 24.0 from C:\Users\adeem\AppData\Local\Programs\Python\Python312\Lib\site-packages\pip (python 3.12)

C:\Users\adeem>pip install pygame
Collecting pygame
  Downloading pygame-2.5.2-cp312-cp312-win_amd64.whl.metadata (13 kB)
Downloading pygame-2.5.2-cp312-cp312-win_amd64.whl (10.8 MB)
   ---------------------------------------- 10.8/10.8 MB 1.8 MB/s eta 0:00:00
Installing collected packages: pygame
Successfully installed pygame-2.5.2

C:\Users\adeem>
```

Una vez que se tienen las maquinas configuradas con la respectiva biblioteca ingresaremos a nuestro símbolo de sistema, e ingresamos el comando ipconfig en el caso de Windows y con ifconfig en el caso de Linux.

```
Sufijo DNS específico para la conexión. . :
Vínculo: dirección IPv6 local. . . : fe80::998d:f3c3:dc9d:e912%27
Dirección IPv4. . . . . . . . . . . . : 192.168.43.137
Máscara de subred . . . . . . . . . . : 255.255.255.0
Puerta de enlace predeterminada . . . . . : 192.168.43.1
```

En este caso es importante que tanto el cliente como el servidor se encuentren conectados a la misma red. Ya que si no, no permitirá la conexión.

Posteriormente, ingresaremos a nuestros códigos, tanto del cliente como el del servidor e ingresaremos el IP correspondiente a la máquina que este fungiendo como servidor.

```python
def main():
    server_ip = '192.168.43.137'
    server_port = 8080
    end_marker = b'EOF'  # Marcador especial para indicar el fin del archivo
```

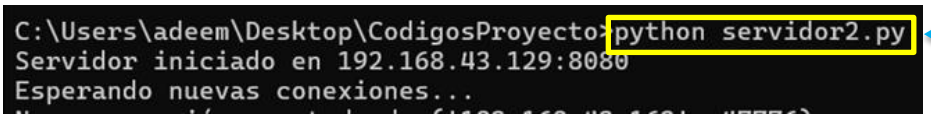
```python
def server():
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        server_socket.bind(('192.168.43.137', 8080))
        server_socket.listen(3)
        print("Servidor iniciado en 192.168.43.137:8080")
    except Exception as e:
        print(f"Error al intentar bind en el puerto: {e}")
        return
```

Los IP se ingresan es estos bloques de código que ya se había explicado anteriormente. Como podemos observar, ambos tienen el mismo.

Una vez que se cambian los IP en nuestros códigos, ingresaremos nuevamente a nuestro símbolo de sistema y ejecutaremos los códigos, ingresando a las carpetas donde los

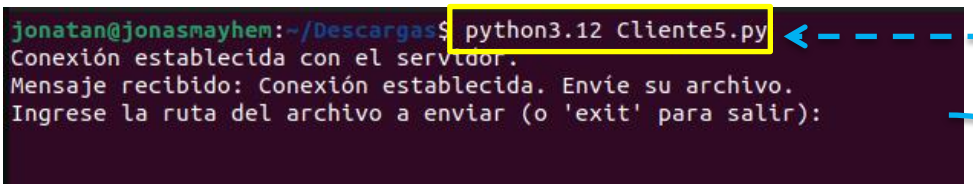
tenemos guardados.

Primero debemos ejecutar al servidor.

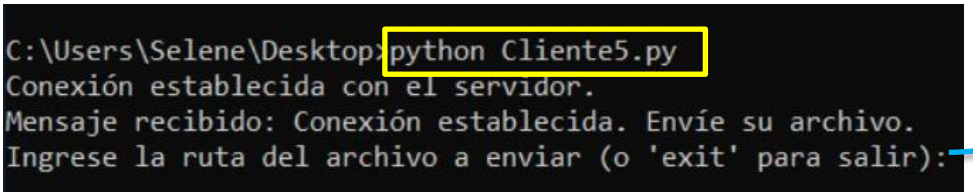


Recordemos que cuando corremos un archivo de Python desde windows, debemos especificar que es pyhton seguido del nombre del archivo con extensión .py

Después de que se inicia el servidor ejecutamos a nuestro cliente o clientes del mismo modo.

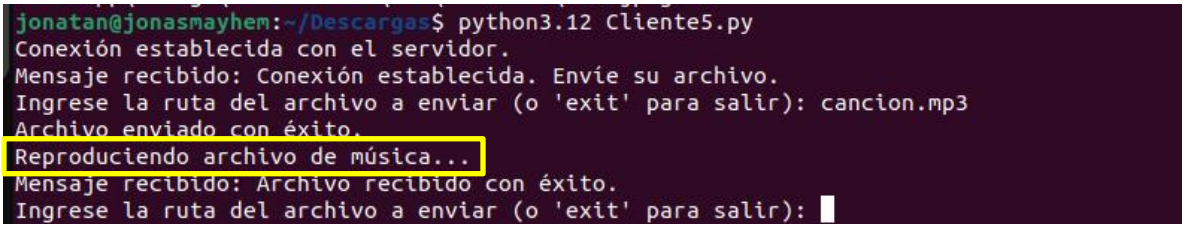


En el caso de Linux, se debe agregar la versión de Python que se tiene instalada.



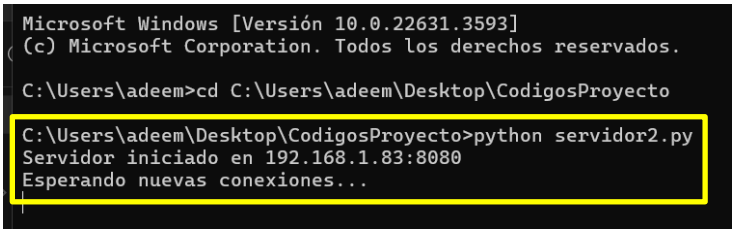
Ambos clientes están en espera de que el usuario ingrese el nombre del archivo.

Finalmente se ingresa el nombre del archivo (nombre de la canción), con extensión .mp3, .wav o .ogg damos enter y el archivo debe comenzar con su reproducción.

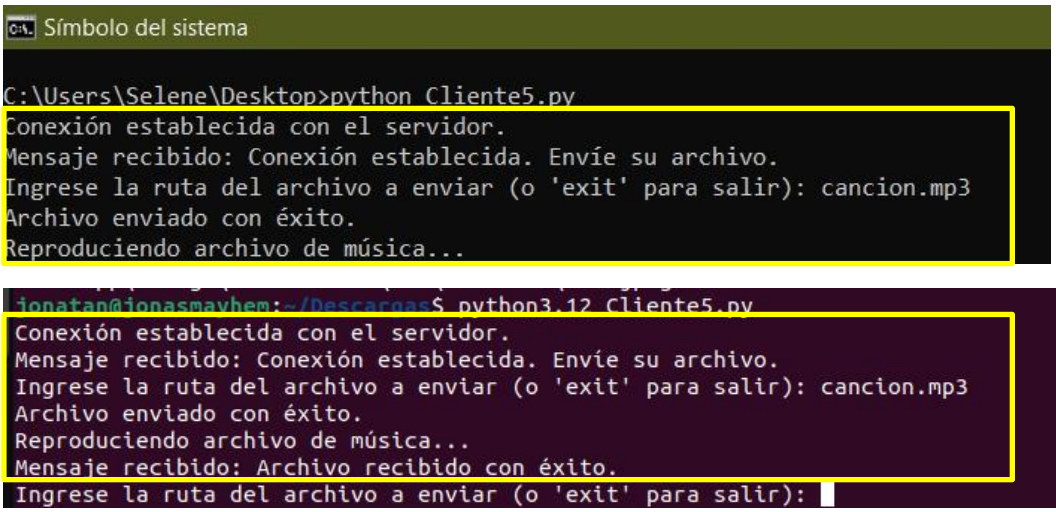


2.4 Resultados

El ejecutar el servidor podemos observar que se conectó correctamente.

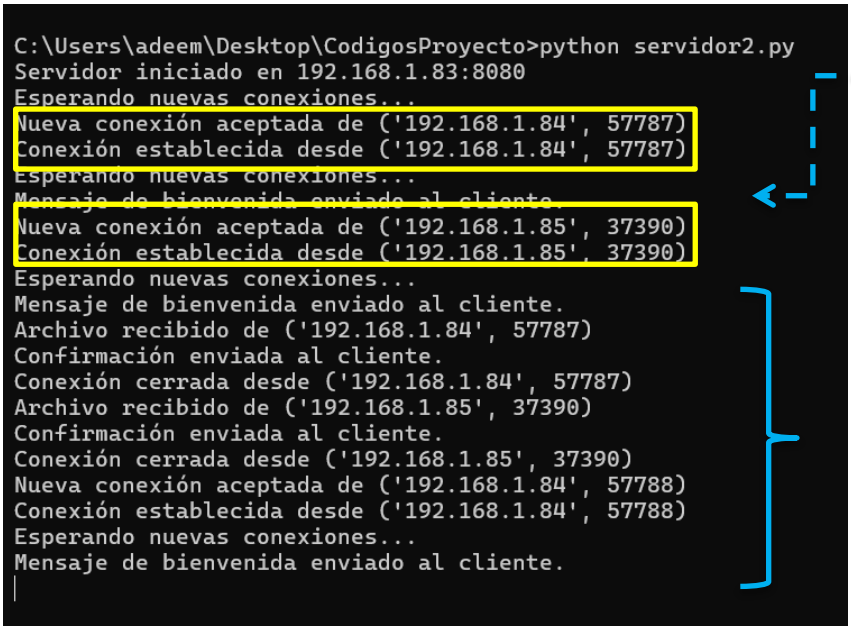


Después ejecutamos al cliente o clientes dependiendo sea el caso



Nota: Para poder ver los resultados finales véase el video adjunto.

Al ejecutar al cliente o los clientes, podemos observar que el servidor recibe las conexiones establecidas desde los diferentes IP.



Diferentes IP de nuestros dos clientes.

Se realizan diferentes pruebas de conectar y desconectar al cliente, en las cuales siempre se recibe respuesta de nueva conexión o conexión cerrada dependiendo sea el caso.

Se cierra la conexión

3. Conclusiones

Raft es un algoritmo de consenso que está diseñado para ser fácil de entender, sin embargo, se debe tener cuidado al momento de implementarlo, ya que puede ser más complejo de lo que parece. Su gran ventaja sobre otros algoritmos es que la comunicación no se corta, a pesar de que alguno de sus clúster pierda la conexión. Como pudimos observar, el experimento no se pudo apreciar del todo bien, ya que solo se contaba con tres máquinas, donde solo dos de ellas estaban correctamente configuradas.

4. Referencia

GitHub. (s.f). El algoritmo de consenso de Raft . Algoritmo de consenso de balsas (raft.github.io)

WikIPedia. (Abril 13, 2024). Balsa (algoritmo). https://en.wikIPedia.org/wiki/Raft_(algorithm)