Informe de Diseño del Sistema Distribuido

1 Arquitectura del Sistema

1.1 Diseño del Sistema

La arquitectura del sistema está basada en una red de tipo **Chord**, donde cada instancia de **Flask** actúa como un nodo. Cada nodo cuenta con las siguientes características:

- **Identificador único**: Generado a partir del hash de su dirección IP y puerto.
- **Tabla de finger**: Utilizada para navegar eficientemente por la red.
- **Base de datos local**: Implementada con **Neo4j**, encargada de almacenar los datos de los que el nodo es responsable.

1.2 Organización del Sistema Distribuido

- **Roles del Sistema**: Cada nodo posee una base de datos local, esencial para garantizar la consistencia de los datos. Además, cada nodo conoce las responsabilidades de los demás nodos en la red, lo que permite una distribución eficiente de tareas.
- **Distribución de Servicios**: Los nodos se encargan de gestionar los servicios de la red social en función de los datos que almacenan. Por ejemplo, si un nodo contiene los datos de los usuarios 1 al 5, será responsable de manejar las operaciones relacionadas con esos usuarios, como seguimientos, publicaciones, etc.

2 Procesos del Sistema

2.1 Servicios de Usuarios

El sistema ofrece una amplia gama de servicios para los usuarios, los cuales se detallan a continuación:

1. **Registro de Usuario** (`/register`, `POST`)

2. **Actualización de Usuario** (`/update-user`, `PUT`)
3. **Eliminación de Usuario** (`/delete-user`, `DELETE`)
4. **Inicio de Sesión de Usuario** (`/login`, `POST`)
5. **Cierre de Sesión de Usuario** (`/logout`, `POST`)
6. **Seguir a un Usuario** (`/follow`, `POST`)
7. **Dejar de Seguir a un Usuario** (`/unfollow`, `POST`)
8. **Buscar Usuarios** (`/find-users`, `GET`)
9. **Crear una Publicación** (`/post`, `POST`)
10. **Repostear una Publicación** (`/repost`, `POST`)
11. **Citar una Publicación** (`/quote`, `POST`)
12. **Eliminar una Publicación** (`/delete-post`, `DELETE`)
13. **Comentar una Publicación** (`/comment-post`, `POST`)
14. **Responder a un Comentario** (`/answer-comment`, `POST`)
15. **Reaccionar a una Publicación** (`/react-post`, `POST`)
16. **Reaccionar a un Comentario** (`/react-comment`, `POST`)
17. **Crear un Gimnasio** (`/create-gym`, `POST`)
18. **Iniciar Sesión como Gimnasio** (`/gym-login`, `POST`)
19. **Actualizar Información del Gimnasio** (`/update-gym`, `PUT`)
20. **Obtener Información del Gimnasio** (`/get-gym-info`, `POST`)
21. **Eliminar Gimnasio** (`/delete-gym`, `DELETE`)
22. **Crear Relación "Entrena en"** (`/trains-in`, `POST`)
23. **Añadir Estilos de Entrenamiento a la Relación** (`/add-training-styles`, `POST`)
24. **Eliminar Estilos de Entrenamiento de la Relación** (`/remove-training-styles`, `POST`)

Cada nodo se encarga de las operaciones relacionadas con los datos que gestiona. Por ejemplo, si un nodo es responsable de los datos de un conjunto de usuarios, manejará todas las operaciones asociadas a esos usuarios, como inicio de sesión, publicaciones, seguimientos, etc.

2.2 Consideraciones sobre Concurrencia

El sistema debe manejar múltiples operaciones simultáneas, lo que implica la necesidad de implementar **hilos** para cada operación. Esto garantiza que los usuarios no tengan que esperar a que se completen las operaciones de otros usuarios. Sin embargo, es crucial implementar mecanismos de **bloqueo** (`lock`) para evitar conflictos en las actualizaciones de la base de datos, especialmente cuando múltiples usuarios intentan modificar la misma información.

3 Comunicación en el Sistema

La comunicación entre nodos se realiza mediante **sockets**, utilizando el protocolo **TCP**, que es orientado a conexión y permite una comunicación bidireccional.

- **Funcionamiento de los Nodos**: Cada nodo actúa como un **servidor** que escucha en un puerto específico para recibir solicitudes de otros nodos. Cuando un nodo necesita obtener información de otro nodo (por ejemplo, encontrar un sucesor o predecesor), actúa como **cliente** y se conecta al servidor del otro nodo.
- **Manejo de Hilos**: Para manejar múltiples operaciones simultáneamente, el sistema utiliza **hilos** (`threading`), lo que permite que cada nodo realice varias tareas en paralelo.

4 Sincronización de Acciones

Cada nodo es responsable de gestionar los servicios relacionados con su segmento de la base de datos. Existe un nodo principal para cada segmento, junto con nodos secundarios que actúan como respaldo. Si el nodo principal falla, se selecciona otro nodo como principal y se reajusta la red.

Las operaciones de escritura en la base de datos deben realizarse con **acceso exclusivo**, asegurando que solo un proceso pueda modificar la base de datos en un momento dado. Aunque cada nodo tiene su propia base de datos local, estas bases de datos están replicadas en otros nodos para garantizar la consistencia. El procedimiento para realizar modificaciones es el siguiente:

1. **Buscar el nodo responsable** de los datos que se desean modificar.
2. **Realizar la modificación** en la base de datos local del nodo principal.
3. **Propagar la modificación** a los nodos secundarios para mantener la consistencia.

Este enfoque minimiza los problemas de **condiciones de carrera**, ya que las operaciones de escritura están sincronizadas.

5 Nombrado y Localización

La distribución de los recursos en la red se realiza de manera automatizada. Cada nodo mantiene una tabla que indica qué recursos gestiona. Cuando se solicita un recurso, el sistema consulta esta tabla para determinar a qué nodo debe dirigirse la solicitud. Si un nodo abandona la red o se incorpora uno nuevo, los recursos se redistribuyen automáticamente.

6 Consistencia y Replicación

La consistencia de los datos se logra mediante la **replicación**. El sistema está diseñado para garantizar que, incluso si se eliminan dos nodos cualesquiera, todos los datos sigan siendo accesibles. Esto se consigue de la siguiente manera:

- Cada nodo es responsable de un conjunto específico de datos (por ejemplo, los datos de los usuarios con ID 1 a 100).
- Además, cada nodo almacena una copia de los datos de los nodos adyacentes (nodo `i-1` y nodo `i+1`).
- De esta forma, si dos nodos fallan, siempre habrá al menos un nodo que contenga la información necesaria.

7 Tolerancia a Fallas

El sistema está diseñado para ser tolerante a fallos. Cada nodo conoce exactamente qué otros nodos contienen los recursos que gestiona. Si el nodo principal falla, se selecciona otro nodo como principal y la red se reajusta para garantizar la consistencia de los datos. Este proceso de estabilización se realiza de manera automática, aunque es importante considerar posibles retrasos (`lag`) para evitar conflictos si un nodo se recupera después de haber sido dado de baja.

La estructura de la red **Chord** facilita esta tolerancia a fallos gracias al uso de la **tabla de finger**, que permite una rápida localización de nodos y recursos.

7.1 Seguridad en la Comunicación

- **Comunicación Segura**: Se implementan mecanismos para asegurar la comunicación entre nodos, como el uso de protocolos cifrados (por ejemplo, **TLS**).

7.2 Seguridad en el Diseño

- **Diseño Seguro**: El sistema incorpora consideraciones de seguridad desde su diseño, incluyendo la validación de entradas y la protección contra ataques comunes.

- **Autorización y Autenticación**: El sistema maneja la autorización y autenticación mediante tokens de acceso y contraseñas cifradas, garantizando que solo los usuarios autorizados puedan realizar operaciones sensibles.